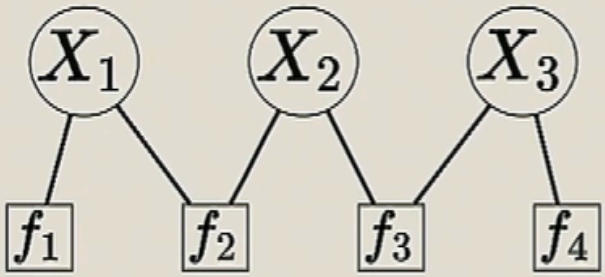
Factor Graphs 2 - Conditional Independence

Review: definition



Definition: factor graph

Variables:

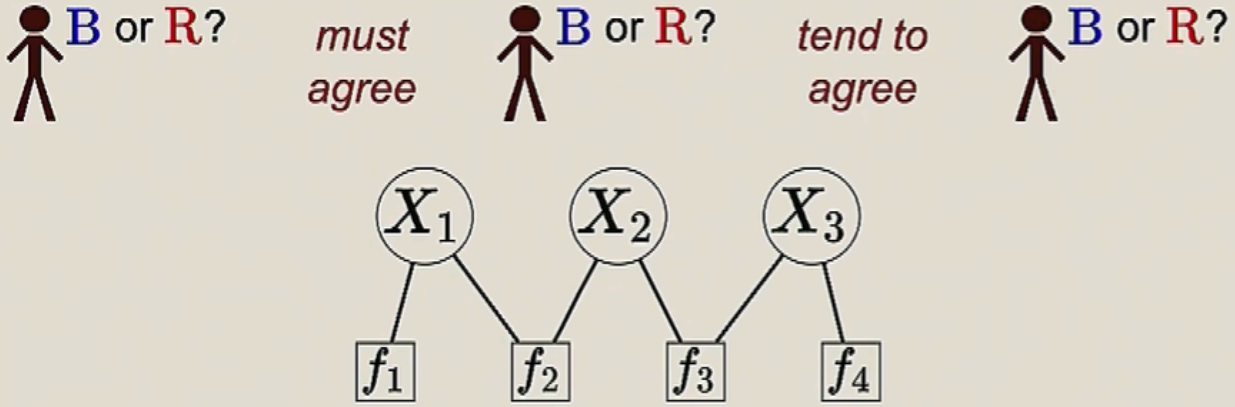
$X = (X_1, \ldots, X_n)$, where $X_i \in \text{Domain}_i$

Factors:

$f_1, \ldots, f_m$, with each $f_j(X) \geq 0$

Scope of f_j: set of dependent variables

Factor graph (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

$$f_2(x_1, x_2) = [x_1 = x_2] \quad f_3(x_2, x_3) = [x_2 = x_3] + 2$$

Review: definition

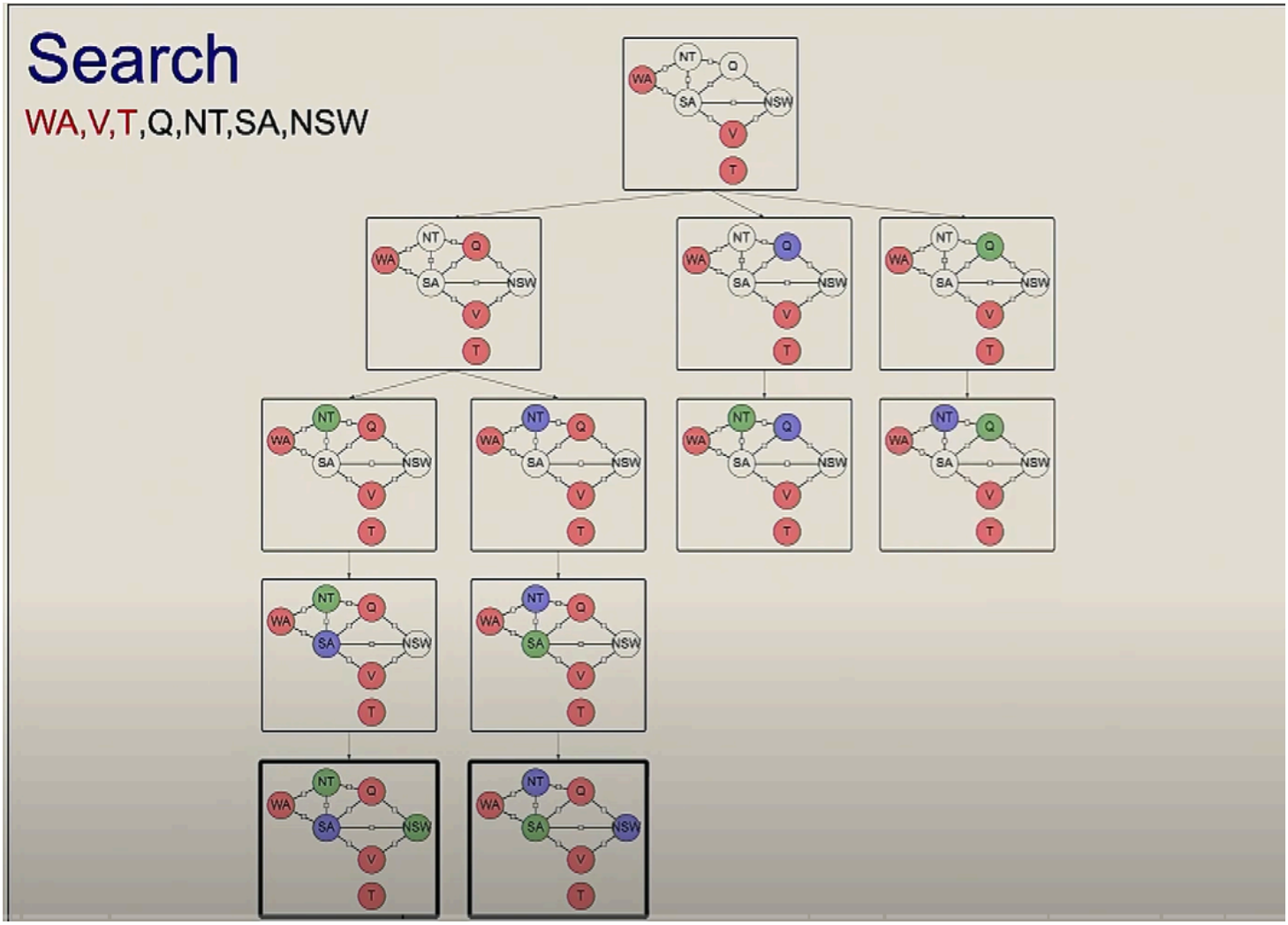
Definition: assignment weight

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

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

Objective: find a maximum weight assignment

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



Gonna be really slow, exponential time

Review: backtracking search

Vanilla version:

$$O(|\text{Domain}|^n) \text{ time}$$

Lookahead: forward checking, AC-3

$$O(|\text{Domain}|^n) \text{ time}$$

Dynamic ordering: most constrained variable, least constrained value

$$O(|\text{Domain}|^n) \text{ time}$$

Note: these pruning techniques useful only for constraints

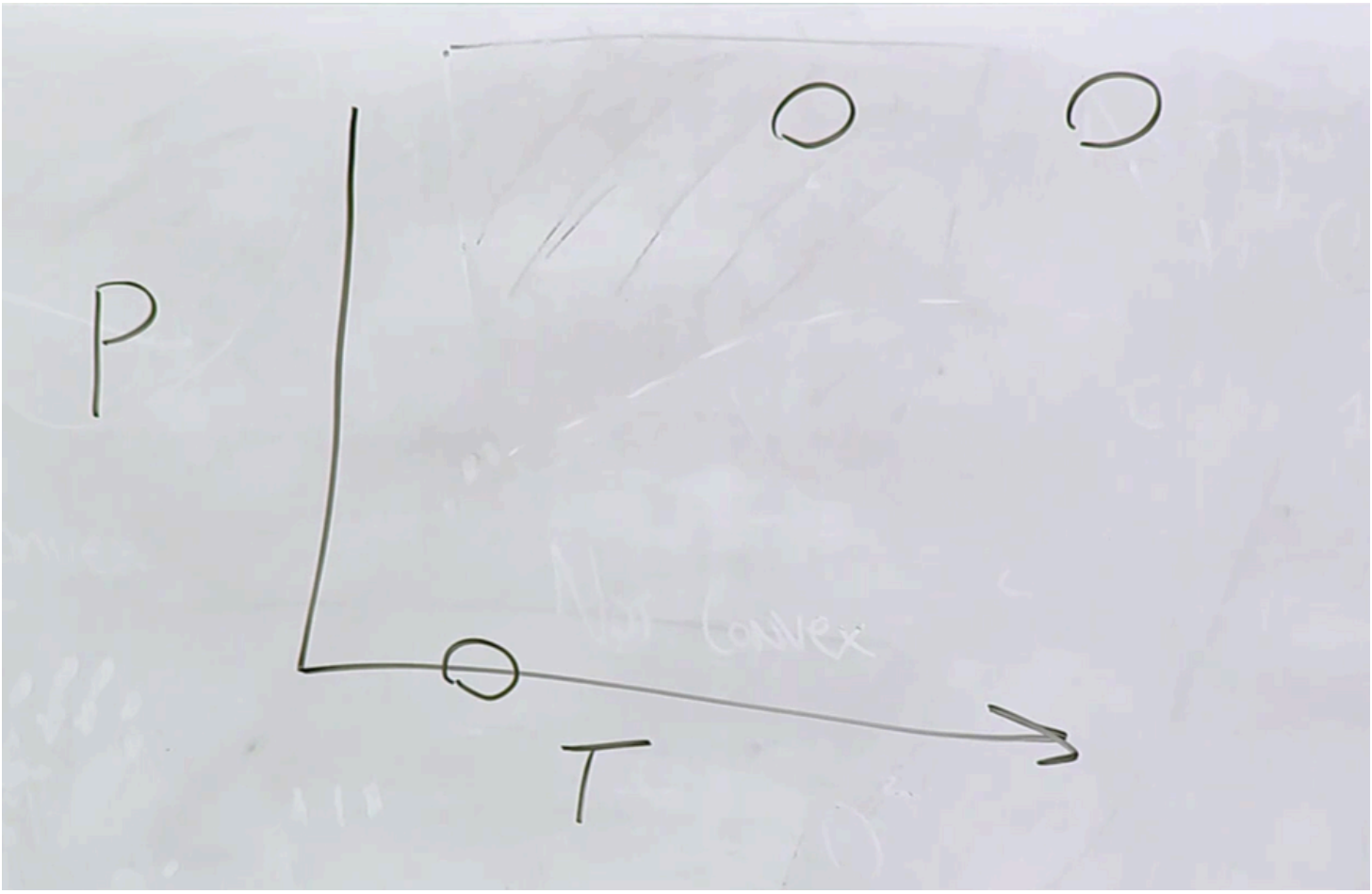
We have these things but they can only work for constraints
only when a factor that gives us a zero, then we know we can prune that branch
So if we have a factor that is non-zero, we can't use any of these...

So, backtracking is slow, but it gives u the optimal solution all the time

Example: object tracking

Setup: sensors (e.g., camera) provide noisy information about location of an object (e.g., video frames)

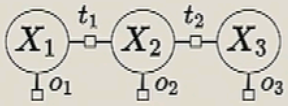
Goal: infer object's true location



We have sensors that gives us noisy estimates for where is the thing every point at a time.
But the sensor is noisy, we cannot fully trust it, so we need to assign true locations

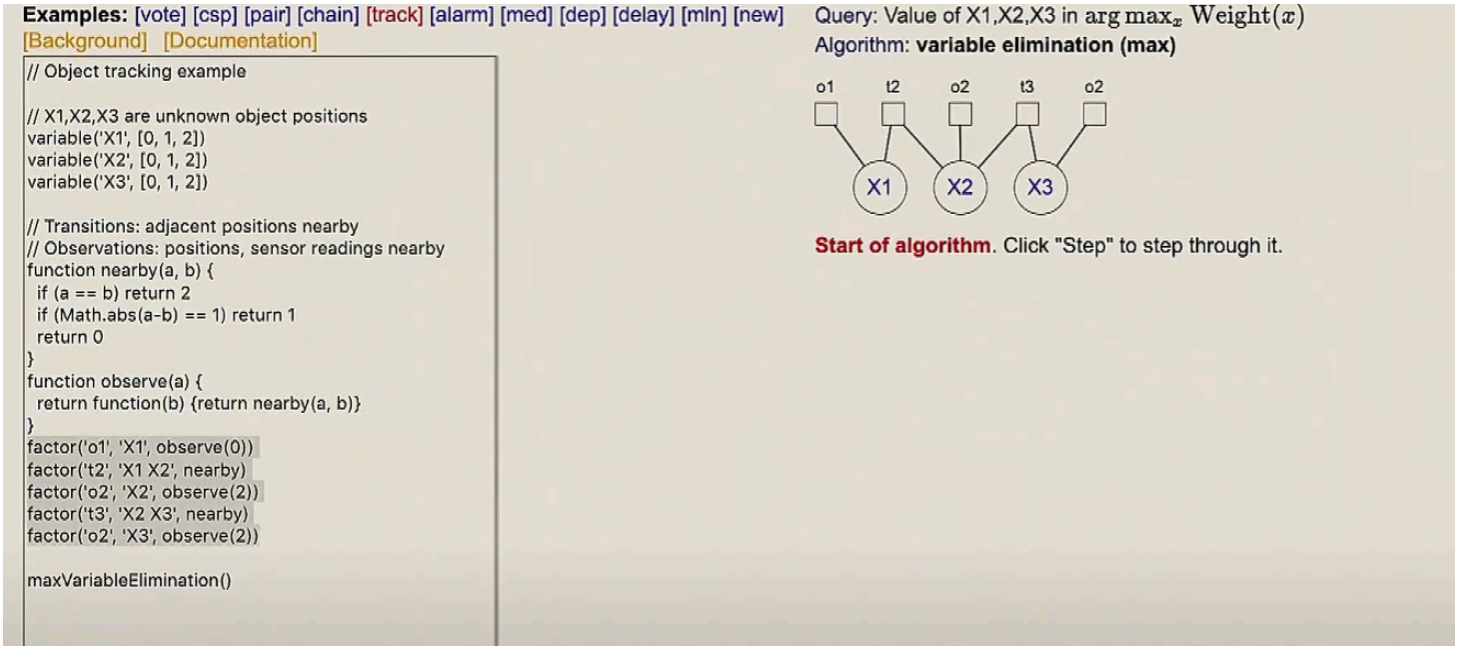
Person tracking solution

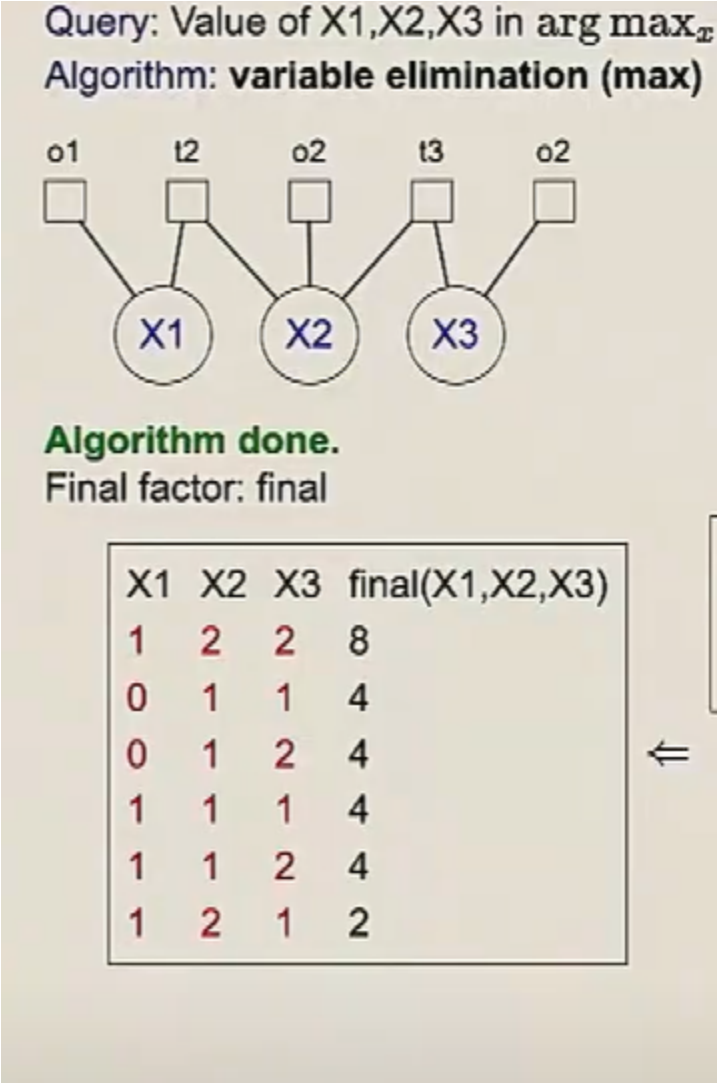
Factor graph (chain-structured):



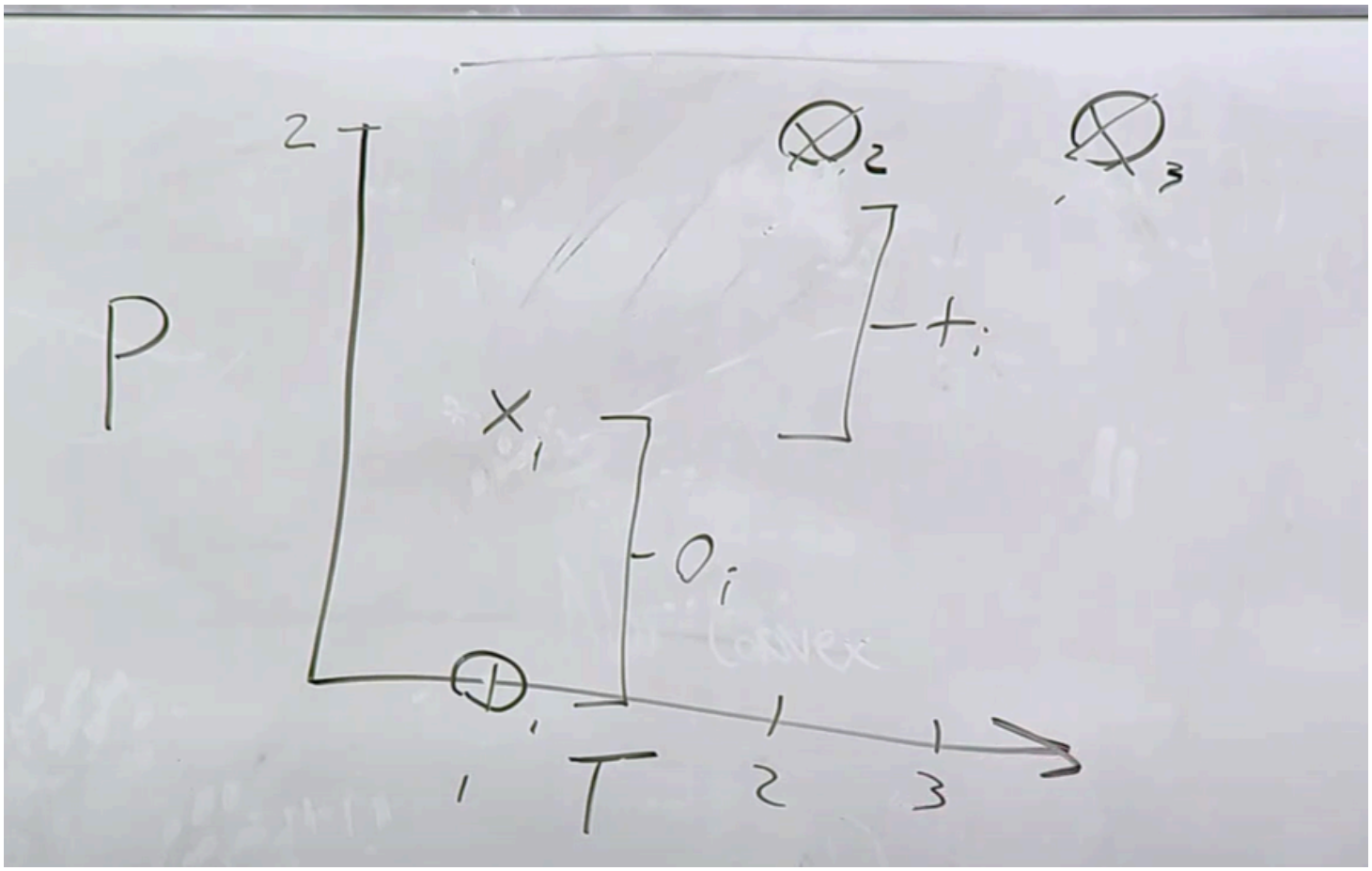
- Variables X_i: location of object at time i
- Observation factors $o_i(x_i)$: noisy information compatible with position
- Transition factors $t_i(x_i, x_{i+1})$: object positions can't change too much

Observation Factor: How close is our guess to the observation(what sensor said)





So now we run the backtrack algorithm, and it returns solution, where X1, X2, X3 are at position 1, 2, 2


P
2
Q_2
Q_3
$+_i$
X_i
O_i
1
T
2
3

Roadmap

Beam search

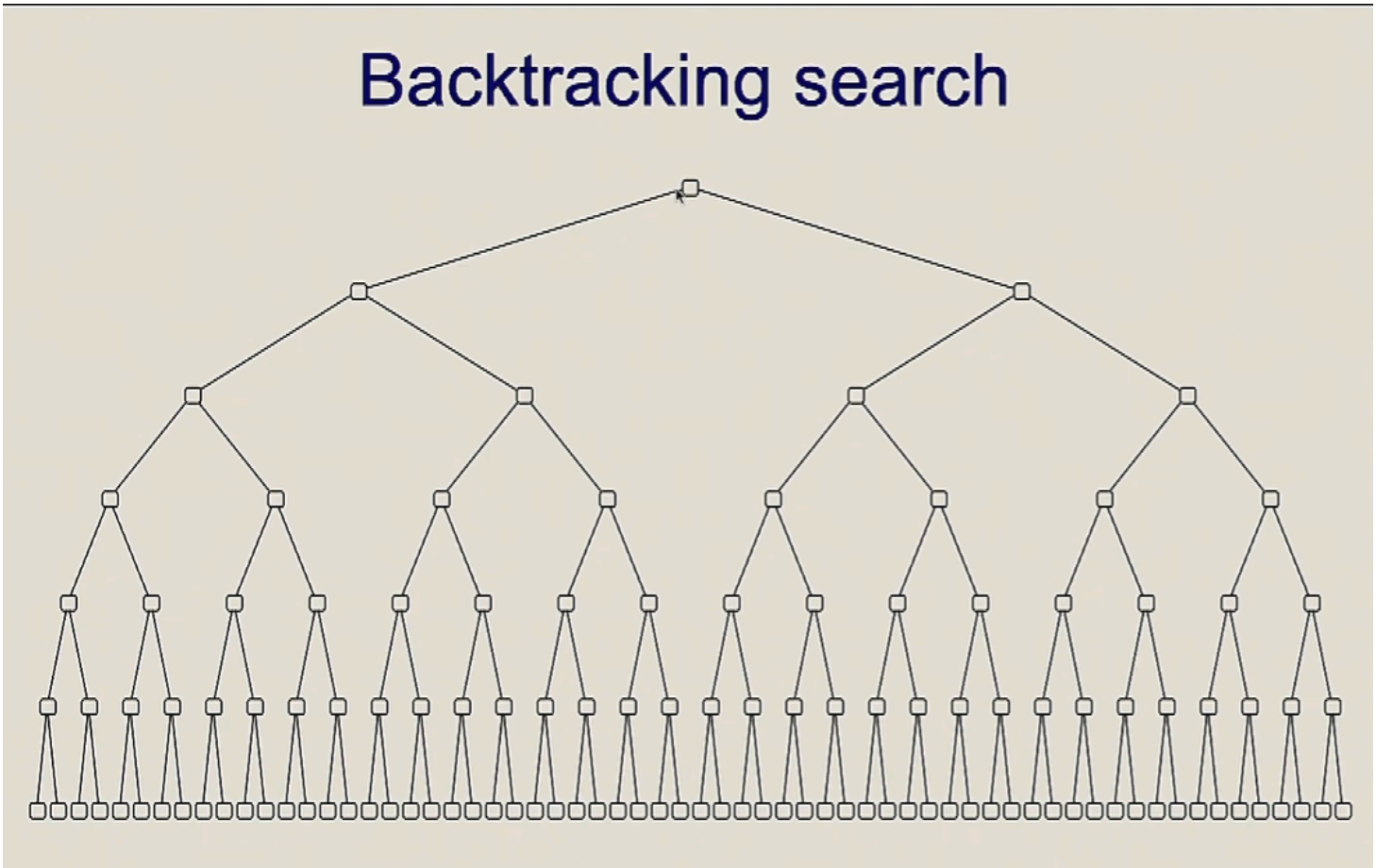
Local search

Conditioning

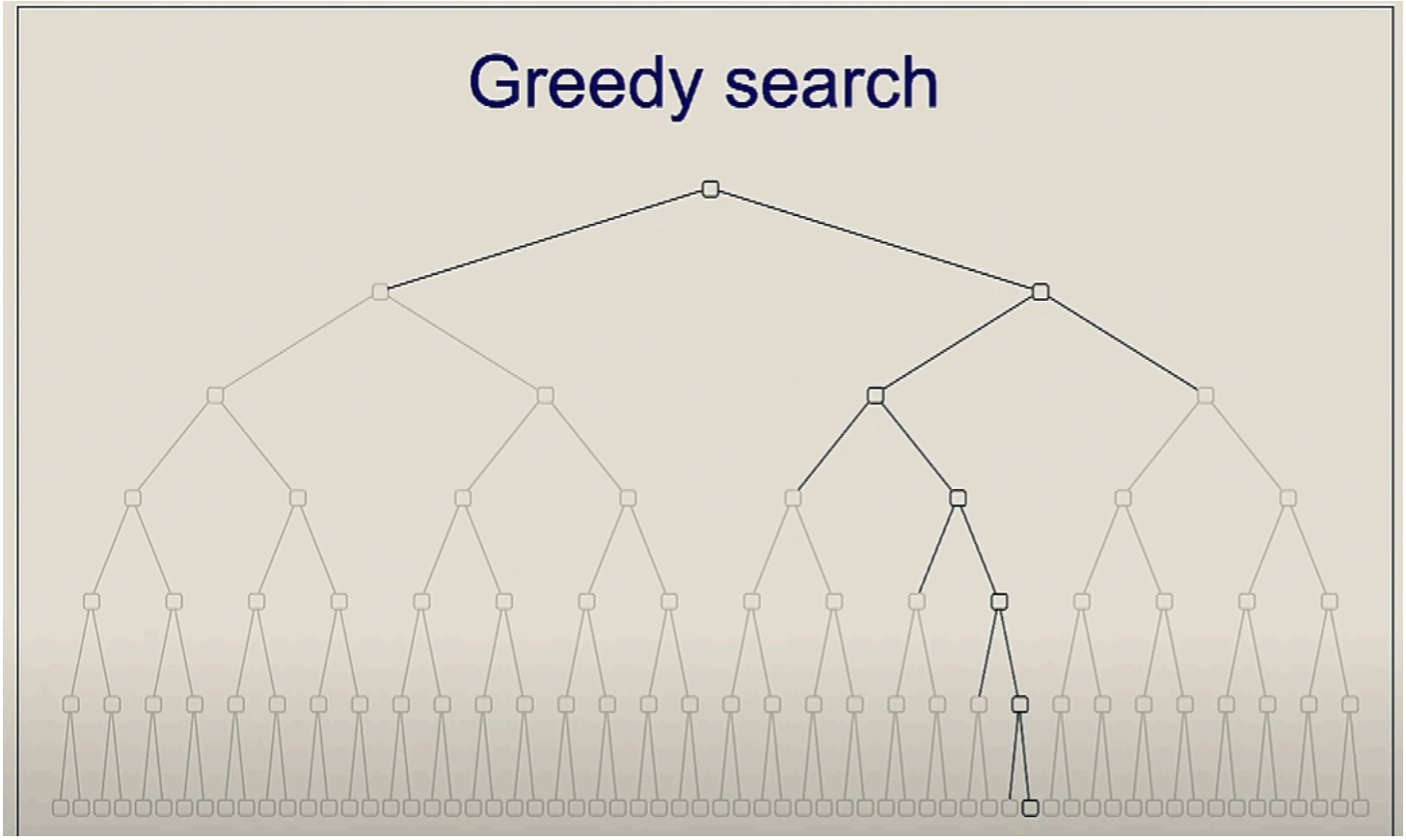
Elimination

Beam Search

Gready search

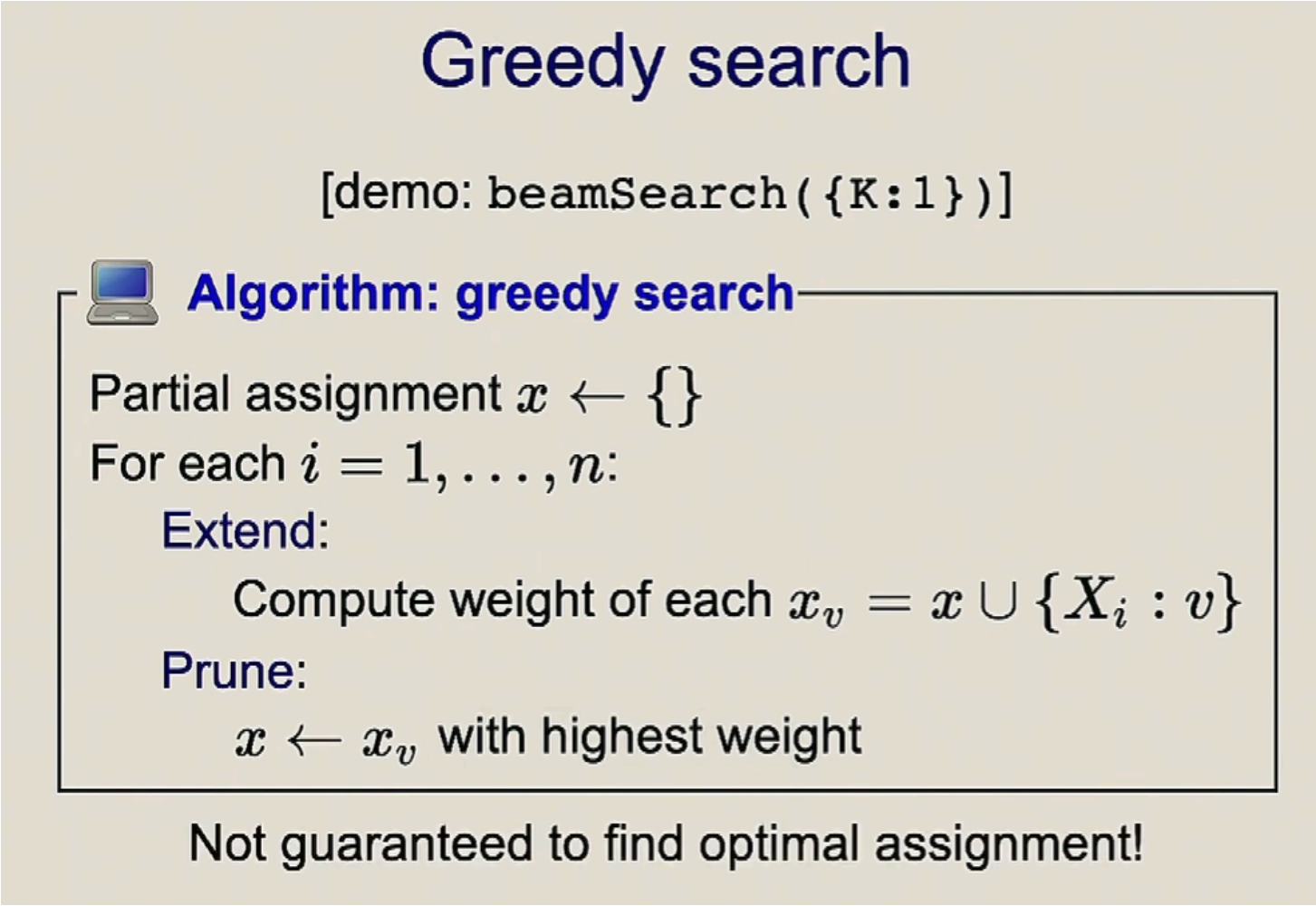


Backtrack gets us the best solution, but it is very slow, let's try to speed it up



One way to speed it up is Greedy search, where we only look at the branch that gives us higher weight at this local window.

Pros: Very fast, linear
Cons: Very narrow window, you don't see a lot of state space, you don't explore so u will often miss the global max



So this basically says we loop and take the highest weight variable only

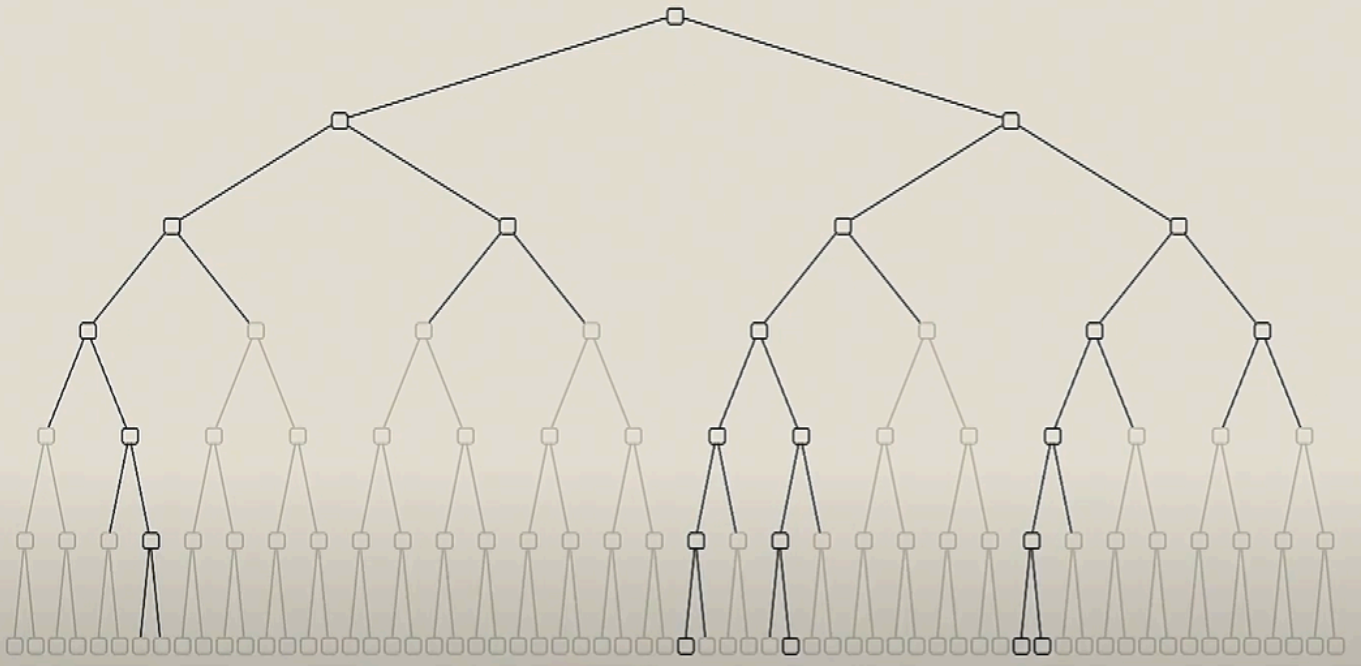
Beam Search

Beam Search is kind of like in between of backtracking and greedy

Instead of maintianing only 1 partial like greedy, we maintian a list of K partial assignments

- We have k partial assignments
- Extend them with **all** possible succssors
- Sort them by their weight and Take only top K partial assignments
- And move on

Beam search



Beam size $K = 4$

Beam search

[demo: `beamSearch({K:3})`]

Idea: keep $\leq K$ **candidate list** C of partial assignments

Algorithm: beam search

Initialize $C \leftarrow [\{\}]$

For each $i = 1, \ldots, n$:

Extend:

$$C' \leftarrow \{x \cup \{X_i : v\} : x \in C, v \in \text{Domain}_i\}$$

Prune:

$C \leftarrow K$ elements of C' with highest weights

Not guaranteed to find optimal assignment!

Beam search properties

- Running time: $O(n(Kb)\log(Kb))$ with branching factor $b = |\text{Domain}|$, beam size K
- Beam size K controls tradeoff between efficiency and accuracy
  - $K = 1$ is greedy ($O(nb)$ time)
  - $K = \infty$ is BFS tree search ($O(b^n)$ time)
- Analogy: backtracking search : DFS :: BFS : beam search (pruned)

Time complexity:

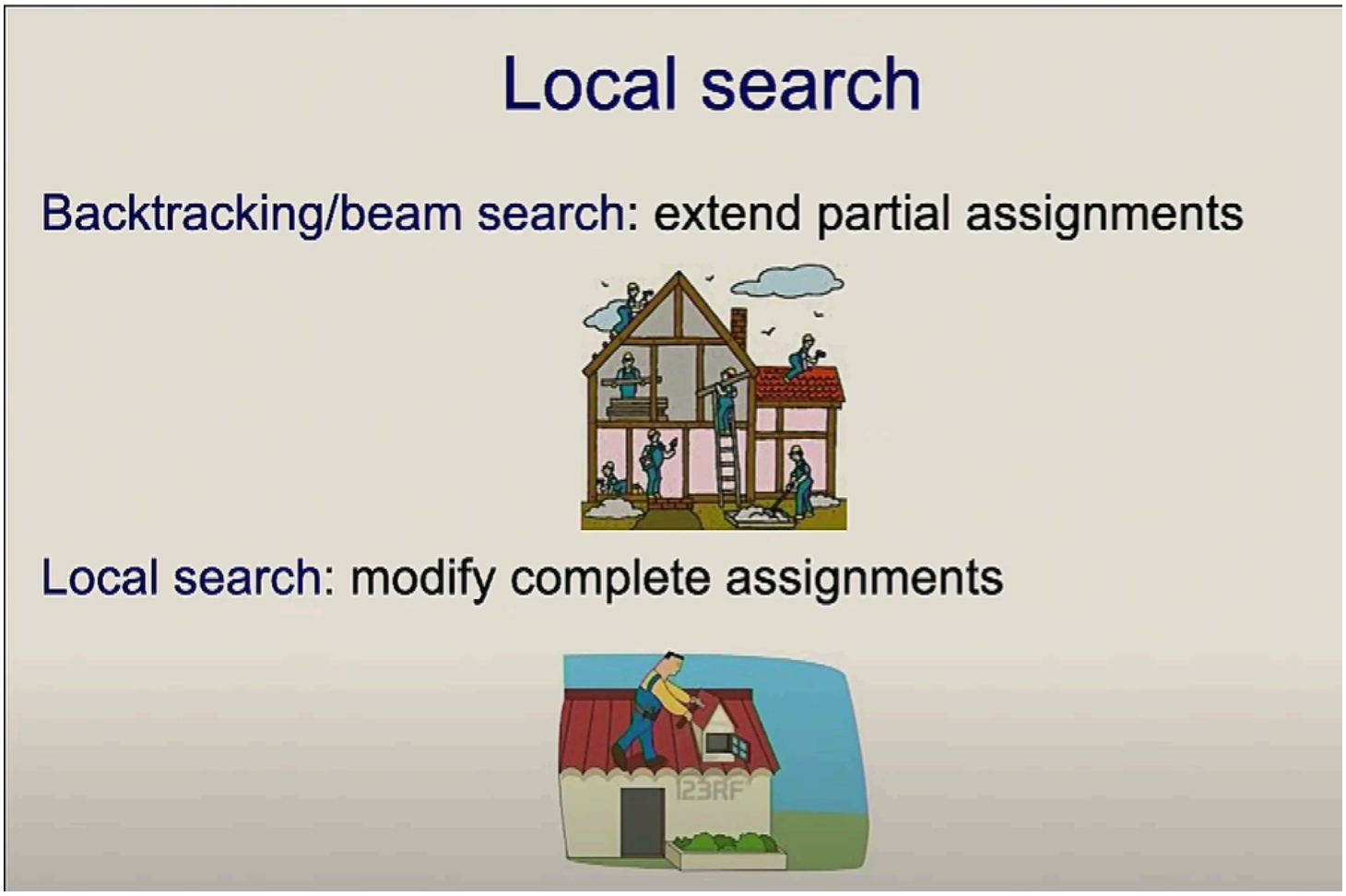
`b` : branching factor - domain, you need to try out every possible solutions

`K` : we are maintaining partial assignments of size K, for each K we try all values in domain. so it's K*b

`(K*b)log(K*b)` : we always need to sort all possible assignments and take top K, the sort takes NlogN

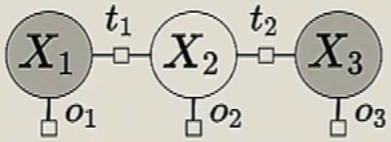
`n` : the depth of the tree - we have n variables

Local search



We already have a complete assignment, and we are trying to make changes to it and improve it (improve the overral weight)

Iterated conditional modes (ICM)



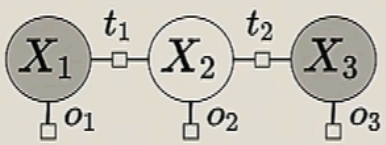
Current assignment: $(0, 0, 1)$; how to improve?

(x_1, v, x_3)	weight
$(0, 0, 1)$	$2 \cdot 2 \cdot 0 \cdot 1 \cdot 1 = 0$
$(0, 1, 1)$	$2 \cdot 1 \cdot 1 \cdot 2 \cdot 1 = 4$
$(0, 2, 1)$	$2 \cdot 0 \cdot 2 \cdot 1 \cdot 1 = 0$

New assignment: $(0, 1, 1)$

In this case we already have the assignment (0,0,1), and improve based on X2
so we try out all the values it can take on, which is 0, 1, 2
Then we recompute the weight and pick whatever value is the best.
In this example we found (0,1,1) is the best

Iterated conditional modes (ICM)



Weight of new assignment (x_1, v, x_3):

$$o_1(x_1)t_1(x_1, v)o_2(v)t_2(v, x_3)o_3(x_3)$$

Key idea: locality

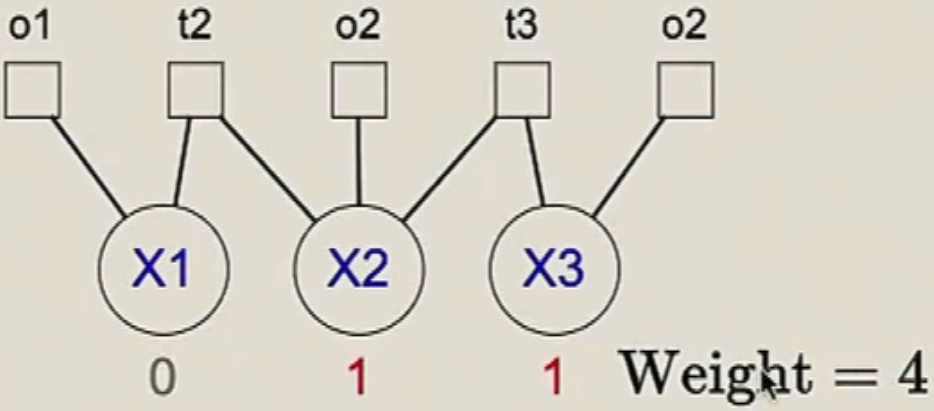
When evaluating possible re-assignments to X_i, only need to consider the factors that depend on X_i.

One cool thing about ICM:
When evaluating a new value for that variable,
we only need to consider factors that touch that variable.
Cuz everything else is consistant with respect to it, and it saves a lot of time.

Iterated conditional modes:
You can solve the whole CSP by this way
But it does not guarentee an optimal solution, it converges to local optimum

Query: Value of X1,X2,X3 in $\arg\max_x \text{Weight}(x)$

Algorithm: **iterated conditional modes (ICM)**



Maximizing variable X1 given everything else:

X1:?	o1	t2	Weight
0	2	1	2
1	1	2	2
2	0	1	0

Choose X1:0

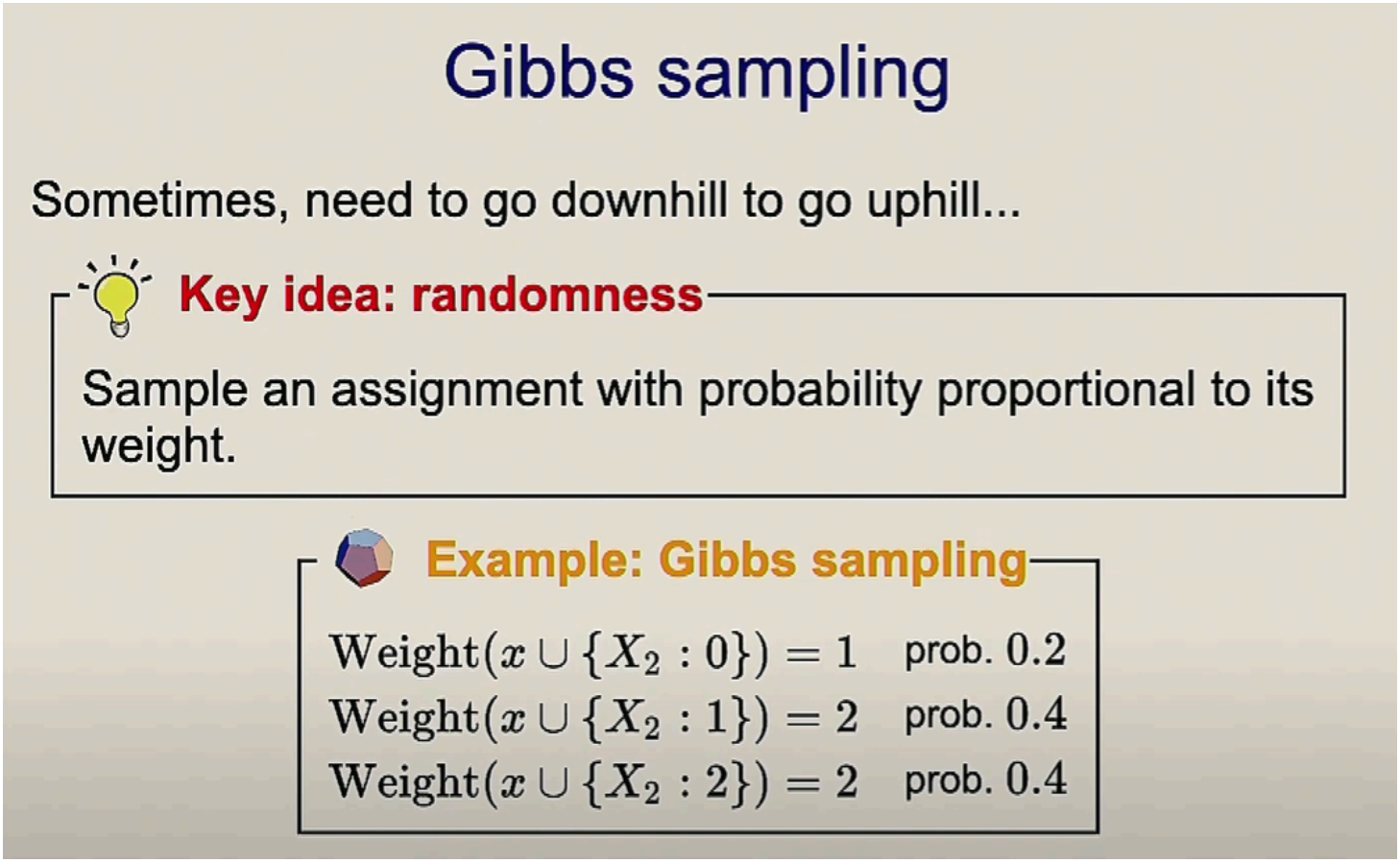
In this example it converges to 4, however backtrack gives us the optimum is 8

- Note that ICM will increase the weight of the assignments monotonically and converges, but it will get stuck in local optima, where there is a better assignment elsewhere, but all the one variable changes result in a lower weight assignment.
- Connection: this hill-climbing is called coordinate-wise ascent. We already saw an instance of coordinate-wise ascent in the K-means algorithm which would alternate between fixing the centroids and optimizing the object with respect to the cluster assignments, and fixing the cluster assignments and optimizing the centroids. Recall that K-means also suffered from local optima issues.
- Connection: these local optima are an example of a Nash equilibrium (for collaborative games), where no unilateral change can improve utility.
- Note that in the demo, ICM gets stuck in a local optimum with weight 4 rather than the global optimum's 8.

Gibbs sampling

One way around this is a second algorithm called `Gibbs sampling`

With Gibbs sampling we are injecting some radomness into the process and try to bump us out of those local optima, into sth that can maybe get up a better area.



Gibbs sampling is super similar to ICM, we still try out all the values

The only difference is that, instead of selecting the value that gives u the heighest weight,

we sample the value, according with probability that's proportional to its weight

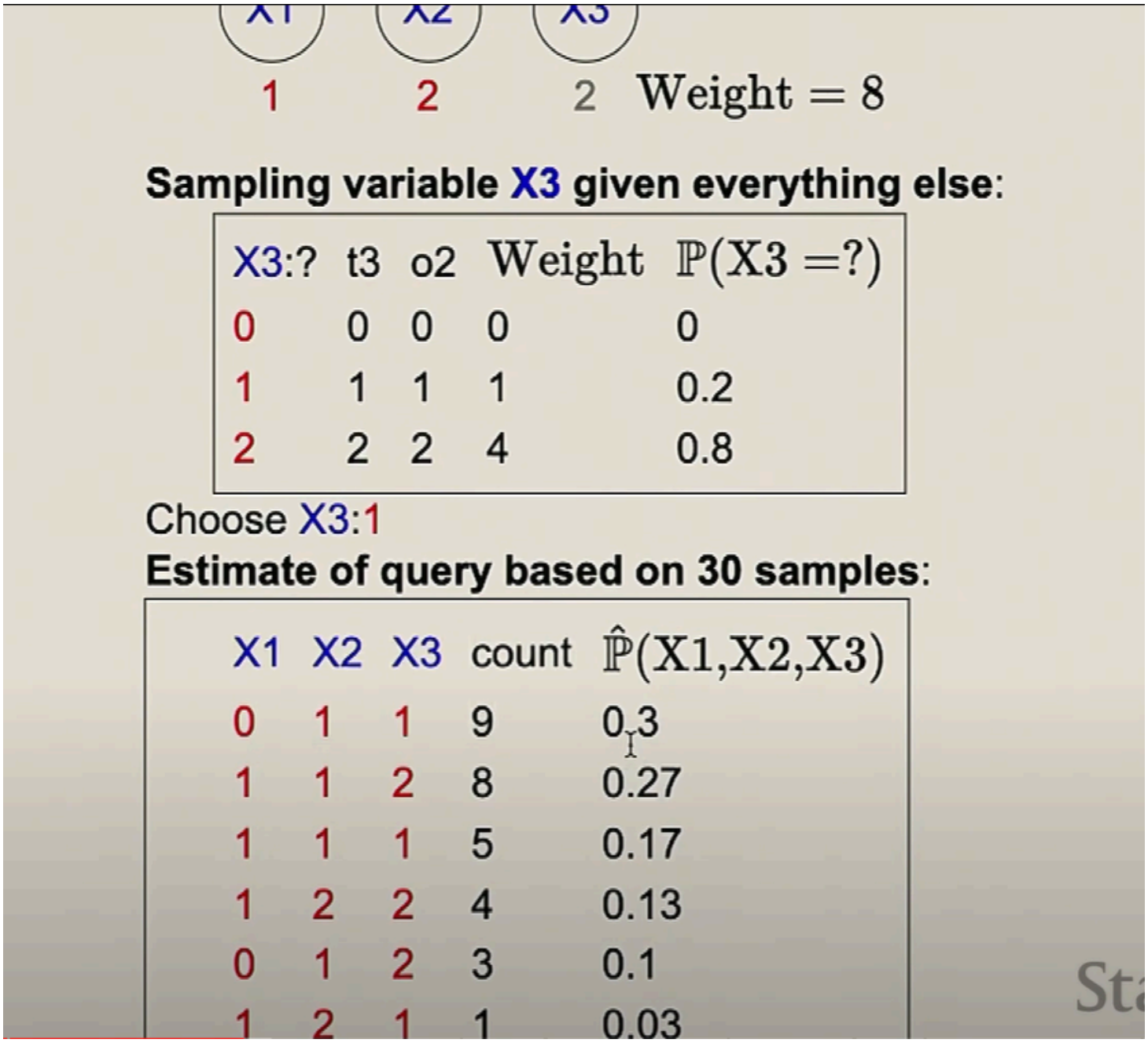
1/5 -> 0.2

2/5 -> 0.4

2/5 -> 0.4

- In reinforcement learning, we also had a problem where if we explore by using a greedy policy (always choosing the best action according to our current estimate of the Q function), then we were doomed to get stuck. There, we used **randomness** via epsilon-greedy to get out of local optima.
- Here, we will do the same, but using a slightly more sophisticated form of randomness. The idea is **Gibbs sampling**, a method originally designed for using Markov chains to sample from a distribution over assignments. We will return to that original use later, but for now, we are going to repurpose it for the problem of finding the maximum weight assignment.

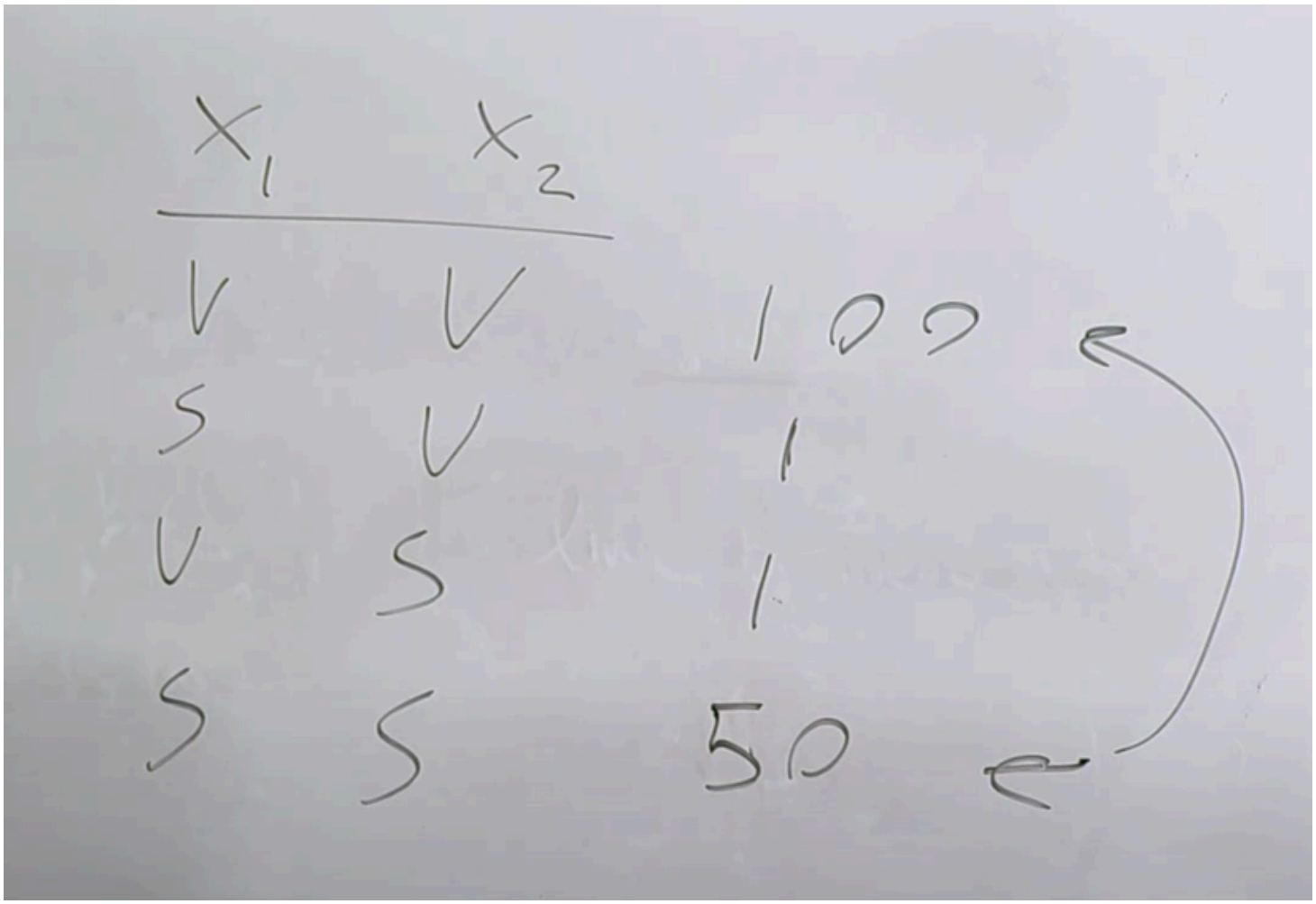
Example



We keep tracking a count of how many times we have this assignment
Overtime, we will find some high weights with some assignment is gonna occur very often

Thus the global optima is (usually) the most frequent, which is really cool

What can go wrong?

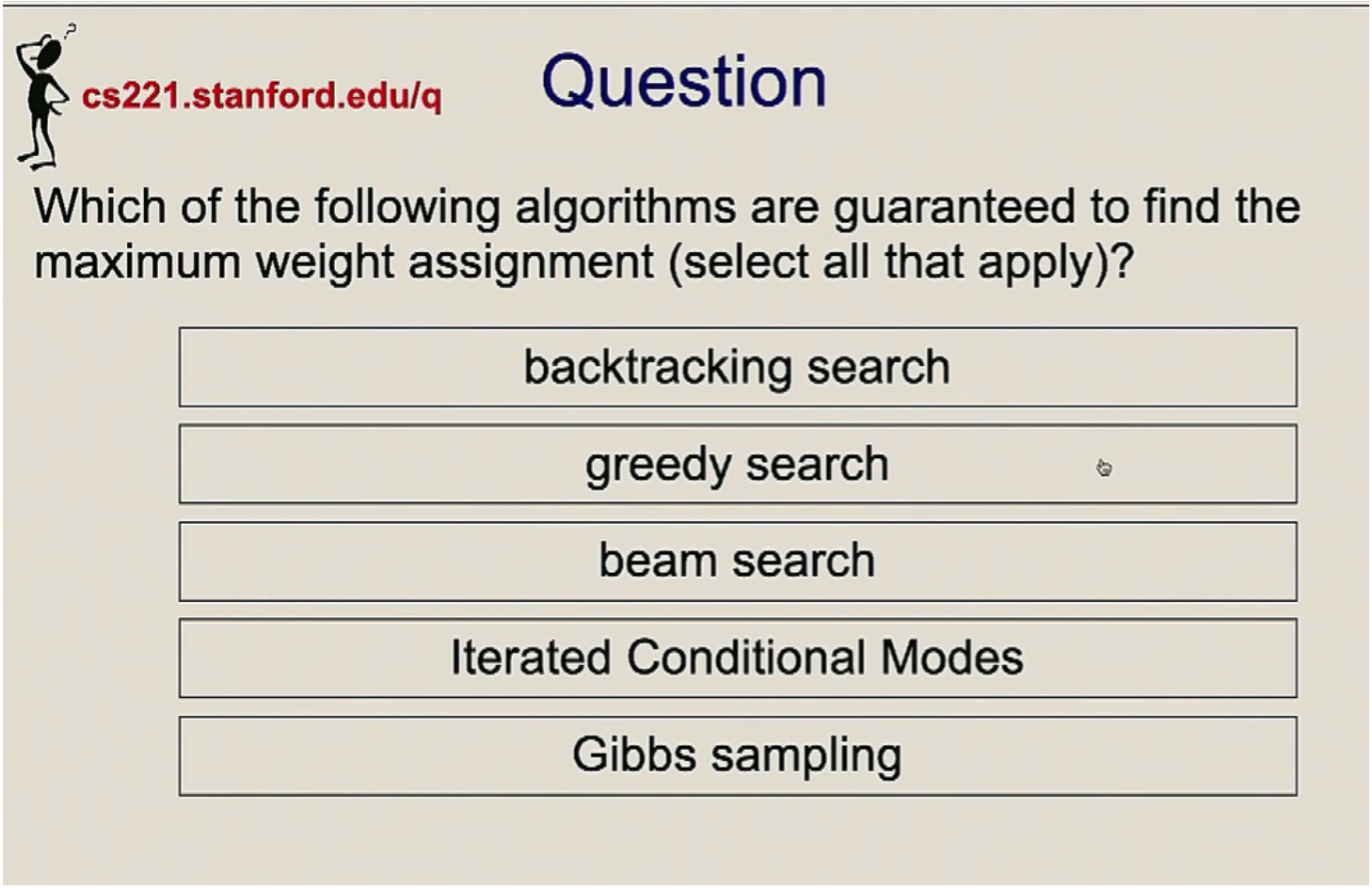


If we have a problem, two ppl, going to restaurant(vege or steak house).
Constraints:
They wanna go to the same restaurant
They both really wanna eat vege.
They eat steaks but they are not super crazy about it

If somehow they follow the probability and choose (S,S)= 50
There's no way they can find the actual global optima is (V,V) = 100
Because changing one variable assignment resulting in (S, V) or (V, S), and it's much worse
It's really hard to bounce between (S,S) and (V,V)

In order to make it to (V,V) from (S,S), one has to make the decision to go to different restaurant, which has a really low value

So a single Gibbs sampling is not guaranteed to find the global optima



Answer: Only backtracking

Greedy is too narrow

Beam search is `maybe` too narrow

ICM is too myopic (short-sighted)

Gibbs sampling is trying to find, it's likely but not a guarantee

Summary so far

Algorithms for max-weight assignments in factor graphs:

(1) Extend partial assignments:

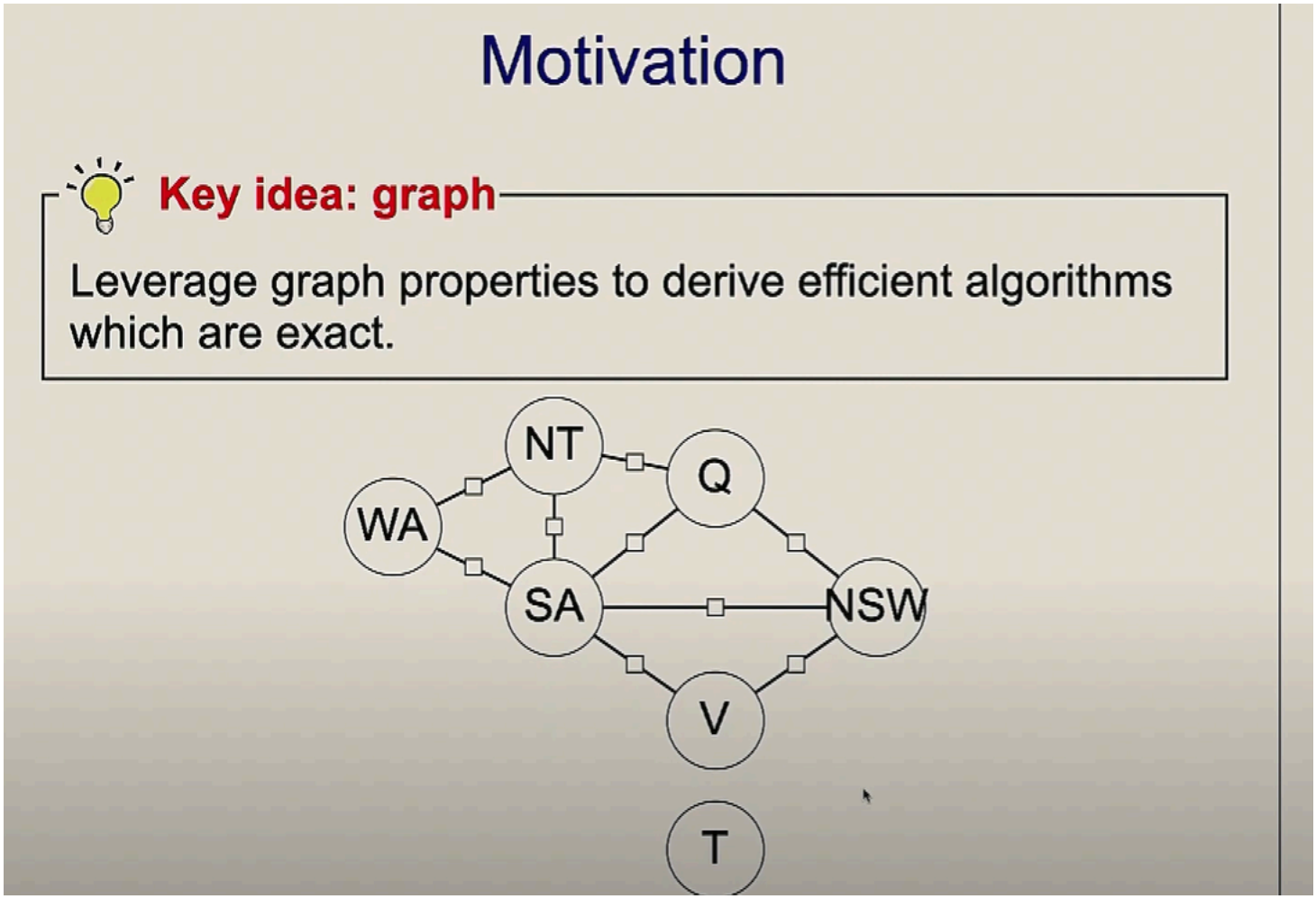
- Backtracking search: exact, exponential time
- Beam search: approximate, linear time

(2) Modify complete assignments:

- Iterated conditional modes: approximate, deterministic
- Gibbs sampling: approximate, randomized

- The goal in the second part of the lecture is to take advantage of the fact that we have a factor **graph**. We will see how exploiting the graph properties can lead us to more efficient algorithms as well as a deeper understanding of the structure of our problem.

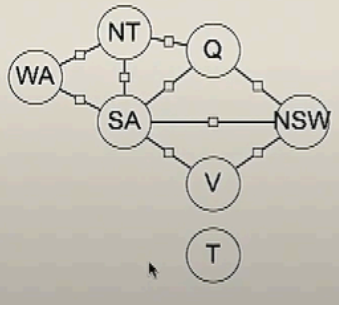
Conditioning



Independence

Definition: independence

- Let A and B be a partitioning of variables X.
- We say A and B are **independent** if there are no edges between A and B.
- In symbols: $A \perp\!\!\!\perp B$.

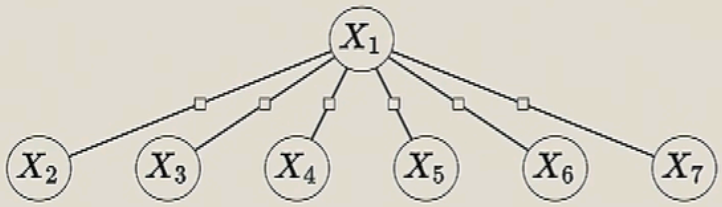


$\{\mathrm{WA}, \mathrm{NT}, \mathrm{SA}, \mathrm{Q}, \mathrm{NSW}, \mathrm{V}\}$ and $\{\mathrm{T}\}$ are independent.

A and B are independent if there no factor/edges/paths that connects them

- Let us formalize this intuition with the notion of **independence**. It turns out that this notion of independence is deeply related to the notion of independence in probability, as we will see in due time.
- Note that we are defining independence purely in terms of the graph structure, which will be important later once we start operating on the graph using two transformations: conditioning and elimination.

Non-independence



No variables are independent of each other, but feels close...

- When all the variables are independent, finding the maximum weight assignment is easily solvable in time linear in n, the number of variables. However, this is not a very interesting factor graph, because the whole point of a factor graph is to model dependencies (preferences and constraints) between variables.
- Consider the tree-structured factor graph, which corresponds to $n - 1$ people talking only through a leader. Nothing is independent here, but intuitively, this graph should be pretty close to independent.

That's when we need to introduce this idea of conditioning

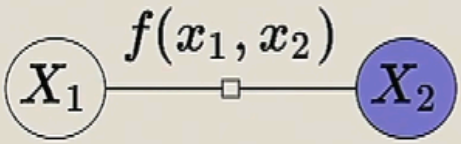
conditioning is a way to rip nodes out of a graph

Q: Will all CSP problems can be represented as graph problems?

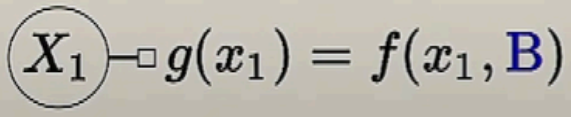
A: Yes, variables are nodes and factors/constraints are edges

Conditioning

Goal: try to disconnect the graph



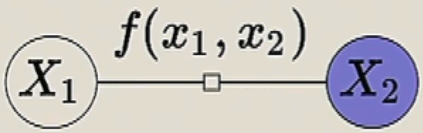
x_1	x_2	$f(x_1, x_2)$
R	R	1
R	B	7
B	R	3
B	B	2



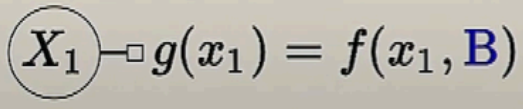
x_1	$g(x_1)$
R	7
B	2

Conditioning

Goal: try to disconnect the graph



x_1	x_2	$f(x_1, x_2)$
R	R	1
R	B	7
B	R	3
B	B	2



x_1	$g(x_1)$
R	7
B	2

Condition on $X_2 = \mathrm{B}$: remove X_2, f and add g

So what if let's say X2 is definitely blue?

We reduced the binary factor into a unary fator(Since X2 doesn't matter now)

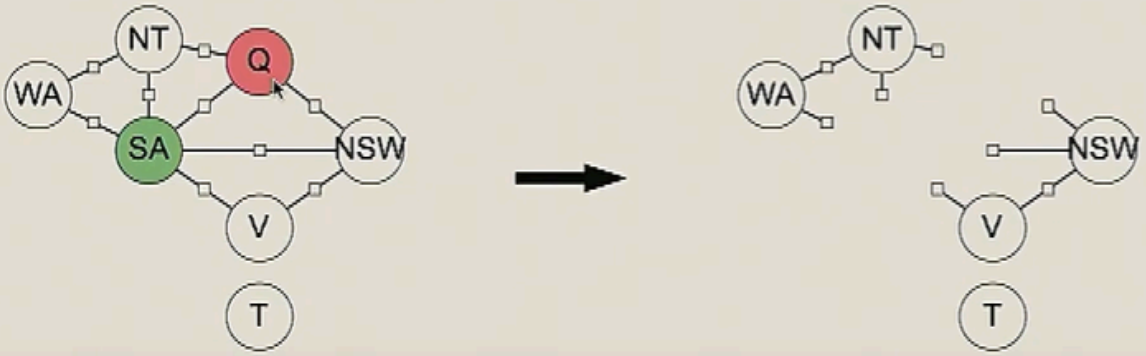
The price to pay is that (at the condition of X2 is being blue)

- In general, factor graphs are not going to have many partitions which are independent (we got lucky with Tasmania, Australia). But perhaps we can transform the graph to make variables independent. This is the idea of **conditioning**: when we condition on a variable $X_i = v$, this is simply saying that we're just going to clamp the value of X_i to v.
- We can understand conditioning in terms of a graph transformation. For each factor f_j that depends on X_i, we create a new factor g_j. The new factor depends on the scope of f_j excluding X_i; when called on x, it just invokes f_j with $x \cup \{X_i : v\}$. Think of g_j as a partial evaluation of f_j in functional programming. The transformed factor graph will have each g_j in place of the f_j and also not have X_i.

Conditioning: example

Example: map coloring

Condition on $\mathrm{Q} = \mathrm{R}$ and $\mathrm{SA} = \mathrm{G}$.



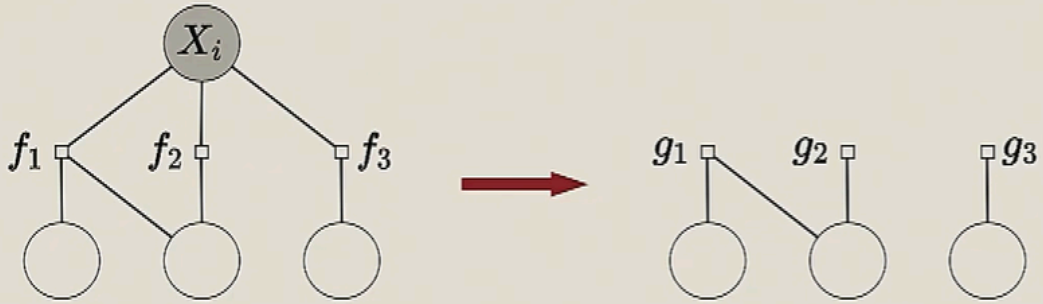
New factors:

$[\mathrm{NT} \neq \mathrm{R}]$
$[\mathrm{NSW} \neq \mathrm{R}]$

$[\mathrm{WA} \neq \mathrm{G}]$
$[\mathrm{NT} \neq \mathrm{G}]$
$[\mathrm{NSW} \neq \mathrm{G}]$
$[\mathrm{V} \neq \mathrm{G}]$

Conditioning: general

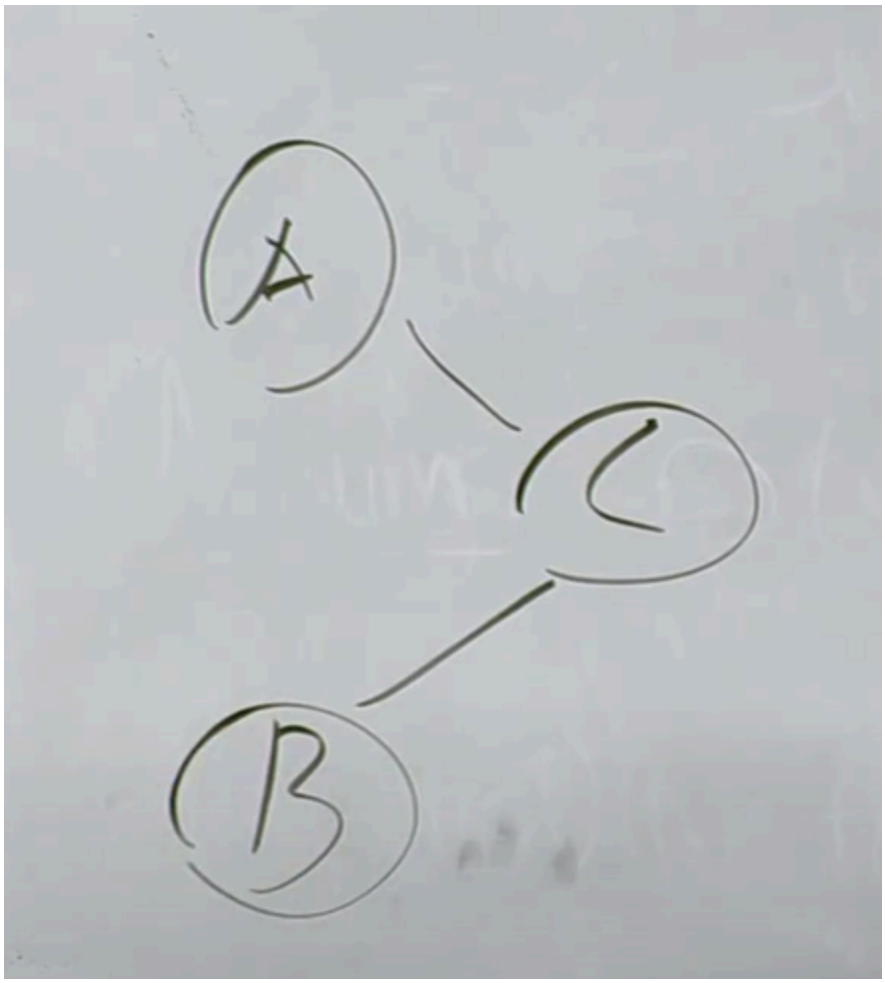
Graphically: remove edges from X_i to dependent factors

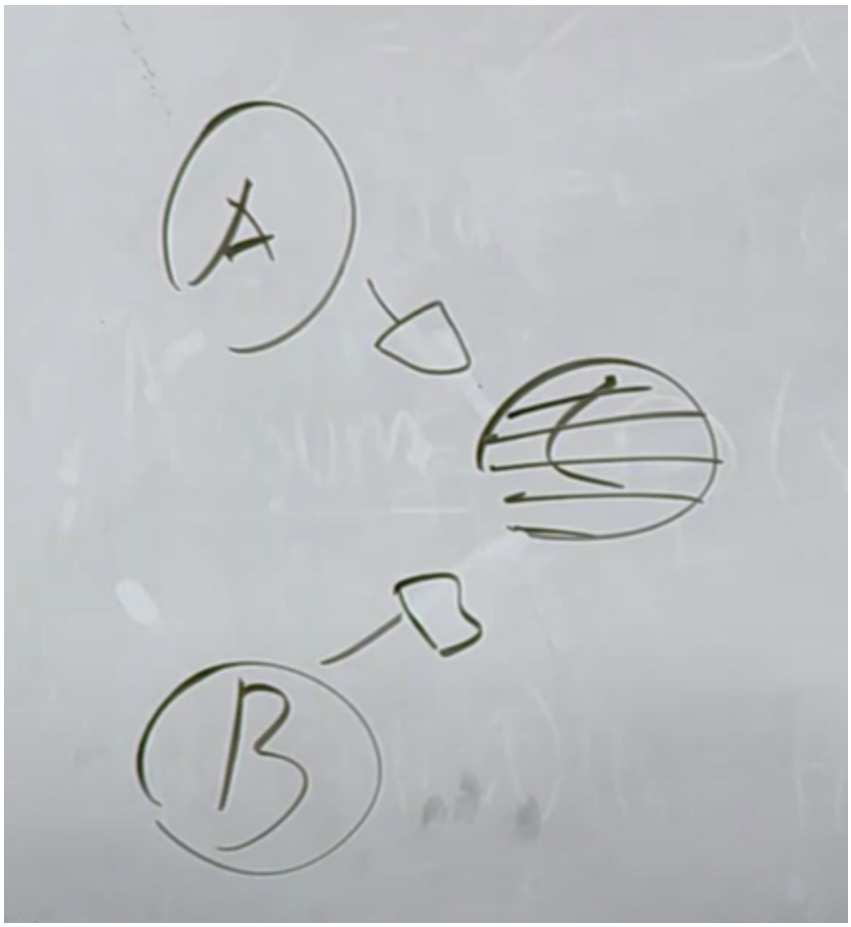


Definition: conditioning

- To **condition** on a variable $X_i = v$, consider all factors $f_1, \ldots, f_k$ that depend on X_i.
- Remove X_i and $f_1, \ldots, f_k$.
- Add $g_j(x) = f_j(x \cup \{X_i : v\})$ for $j = 1, \ldots, k$.

Conditional independent





We say A and B are conditionally independent, because once we condition on C(once we pick a value for C),
A - C and B - C turns into stumps, and now there's no edges or path that connecting them or reach each other.

Conditional independence

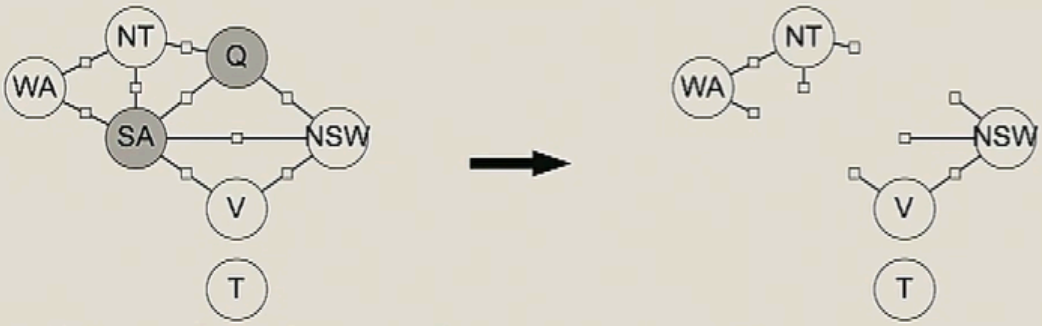
Definition: conditional independence

- Let A, B, C be a partitioning of the variables.
- We say A and B are **conditionally independent** given C if conditioning on C produces a graph in which A and B are independent.
- In symbols: $A \perp\!\!\!\perp B \mid C$.

Equivalently: every path from A to B goes through C.

Conditional independence

Example: map coloring



Conditional independence assertion:

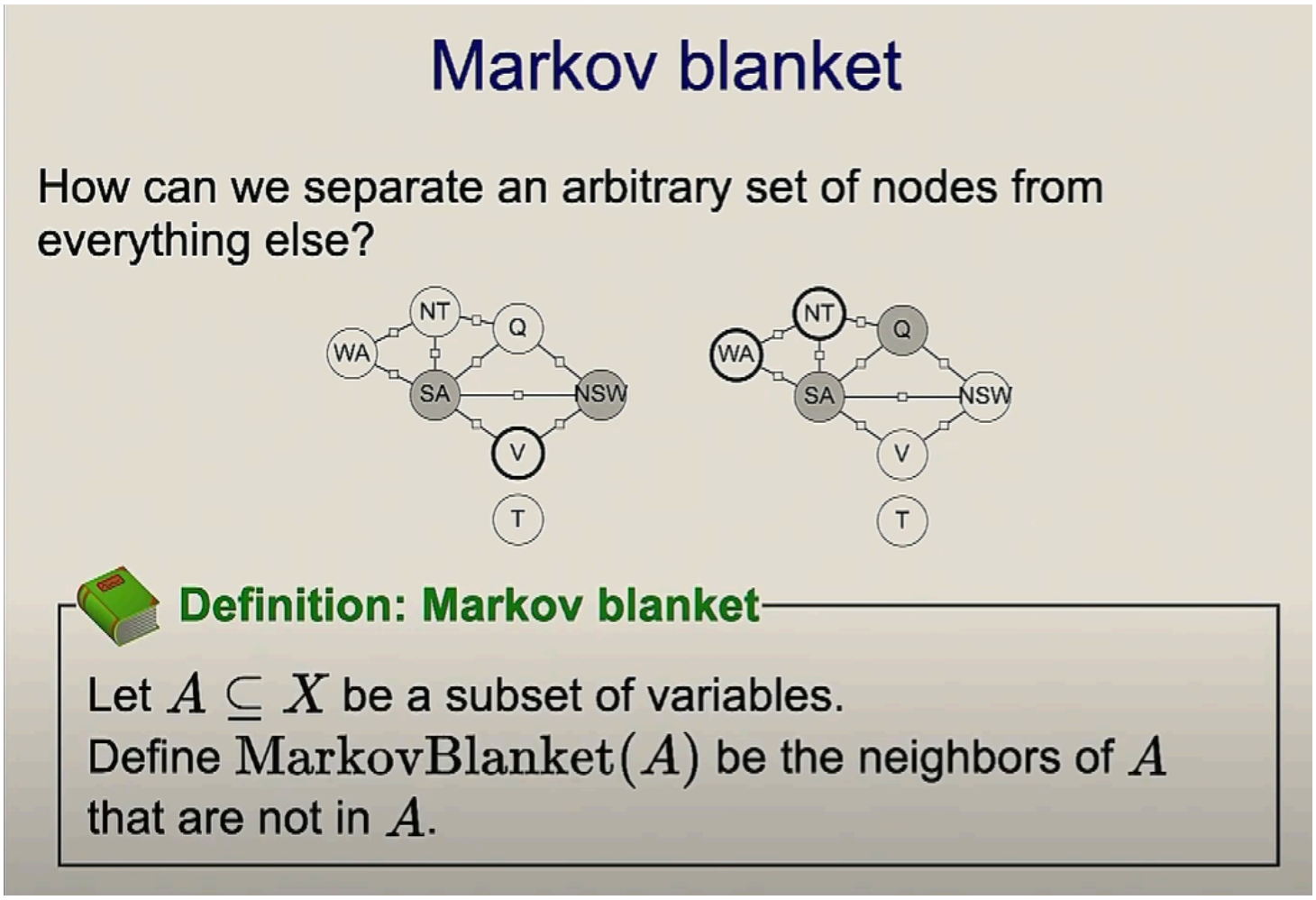
$$\{\mathrm{WA}, \mathrm{NT}\} \perp\!\!\!\perp \{\mathrm{V}, \mathrm{NSW}, \mathrm{T}\} \mid \{\mathrm{SA}, \mathrm{Q}\}$$

Markov blanket

Pick one or more variables

What are the variables that I have to destory to make my variable an island(independent)

Examples:

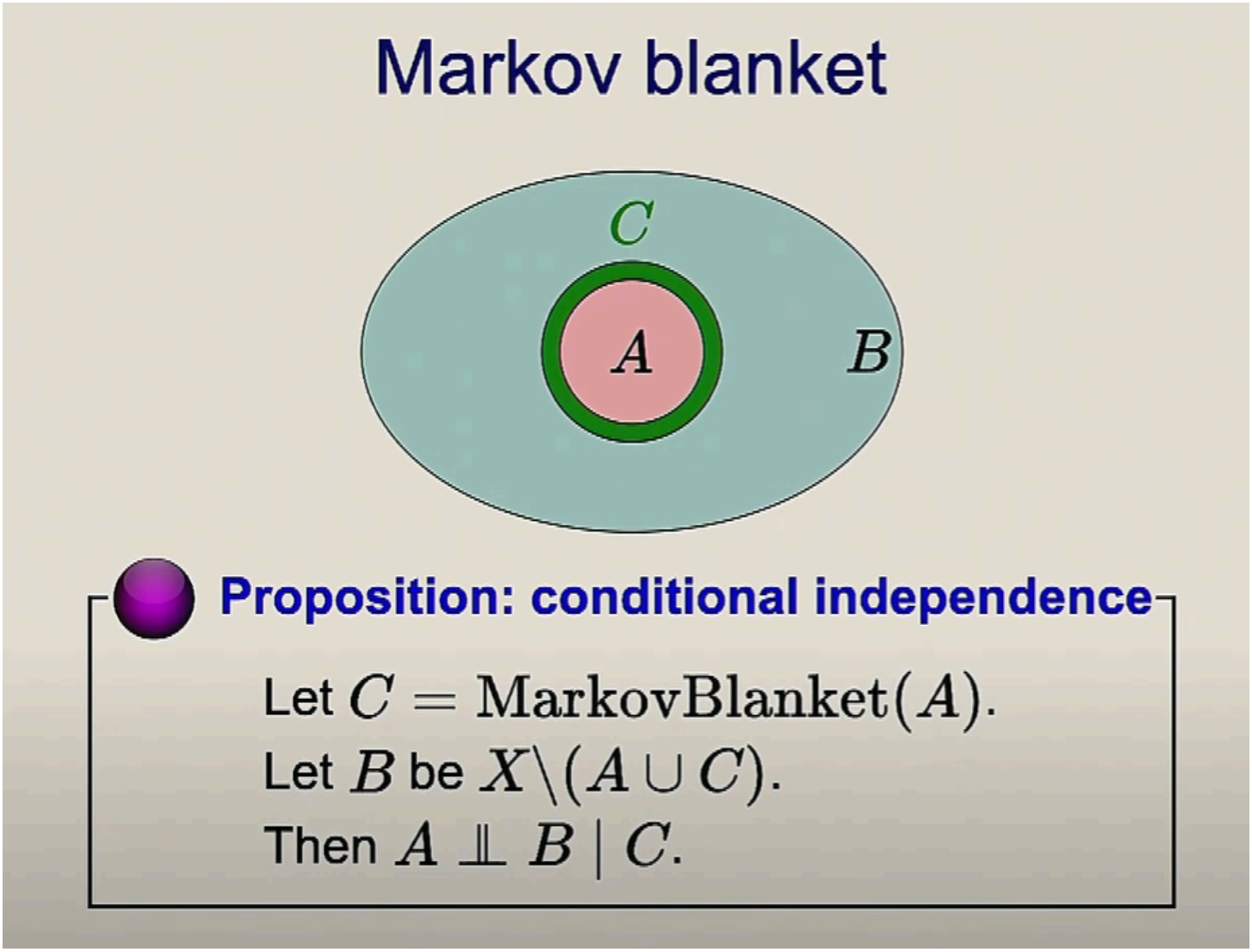


Pick: V

Markov blanket: SA NSW

The set of node you have to conditional on, is called the markov blanket of the set of nodes you want to be independent

The set notion:

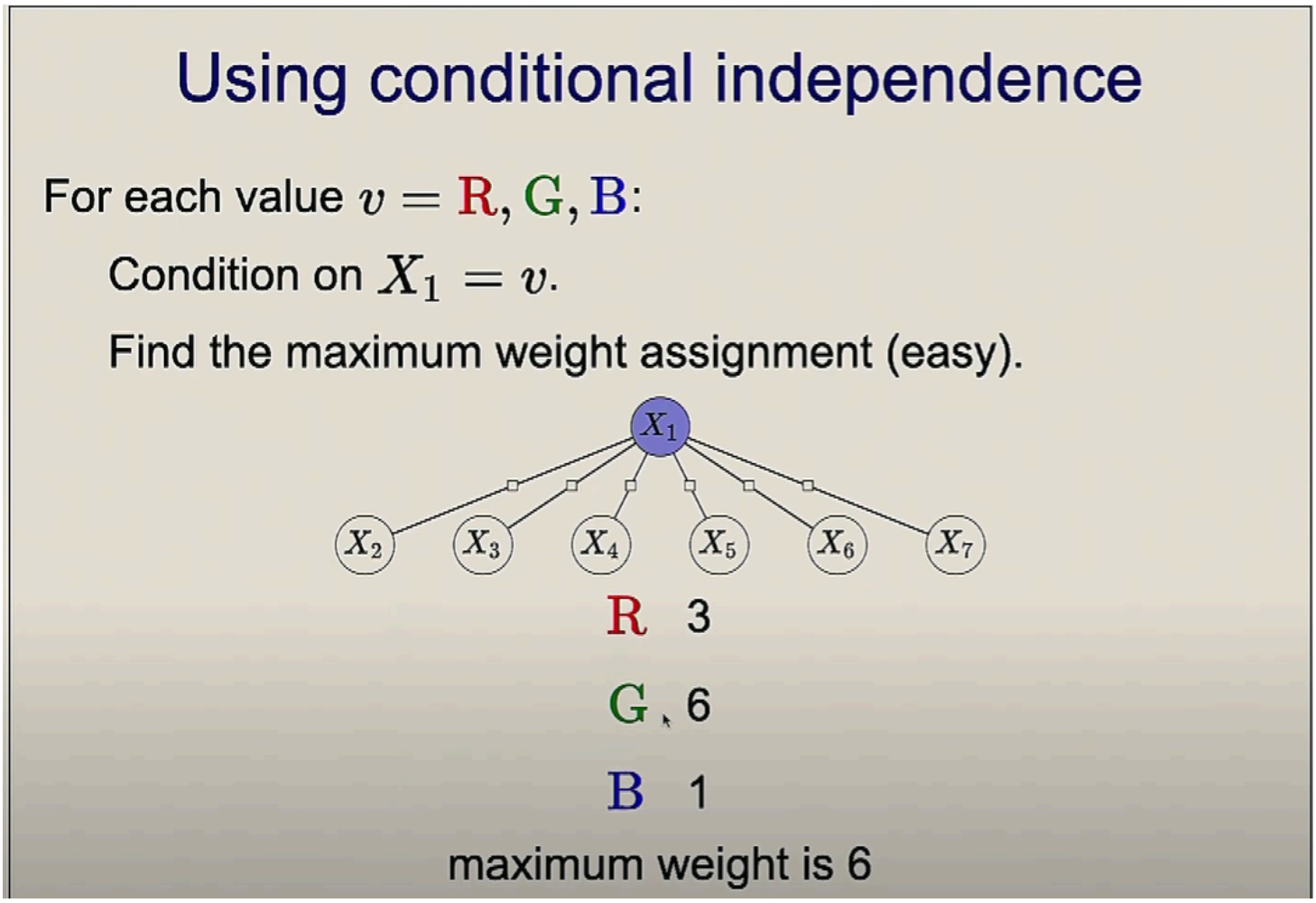


- Suppose we wanted to disconnect a subset of variables $A \subset X$ from the rest of the graph. What is the smallest set of variables C that we need to condition on to make A and the rest of the graph ($B = X \backslash (A \cup C)$) conditionally independent.
- It's intuitive that the answer is simply all the neighbors of A (those that share a common factor) which are not in A. This concept is useful enough that it has a special name: **Markov blanket**.
- Intuitively, the smaller the Markov blanket, the easier the factor graph is to deal with.

Example

So now we can use this idea to create independent structure now

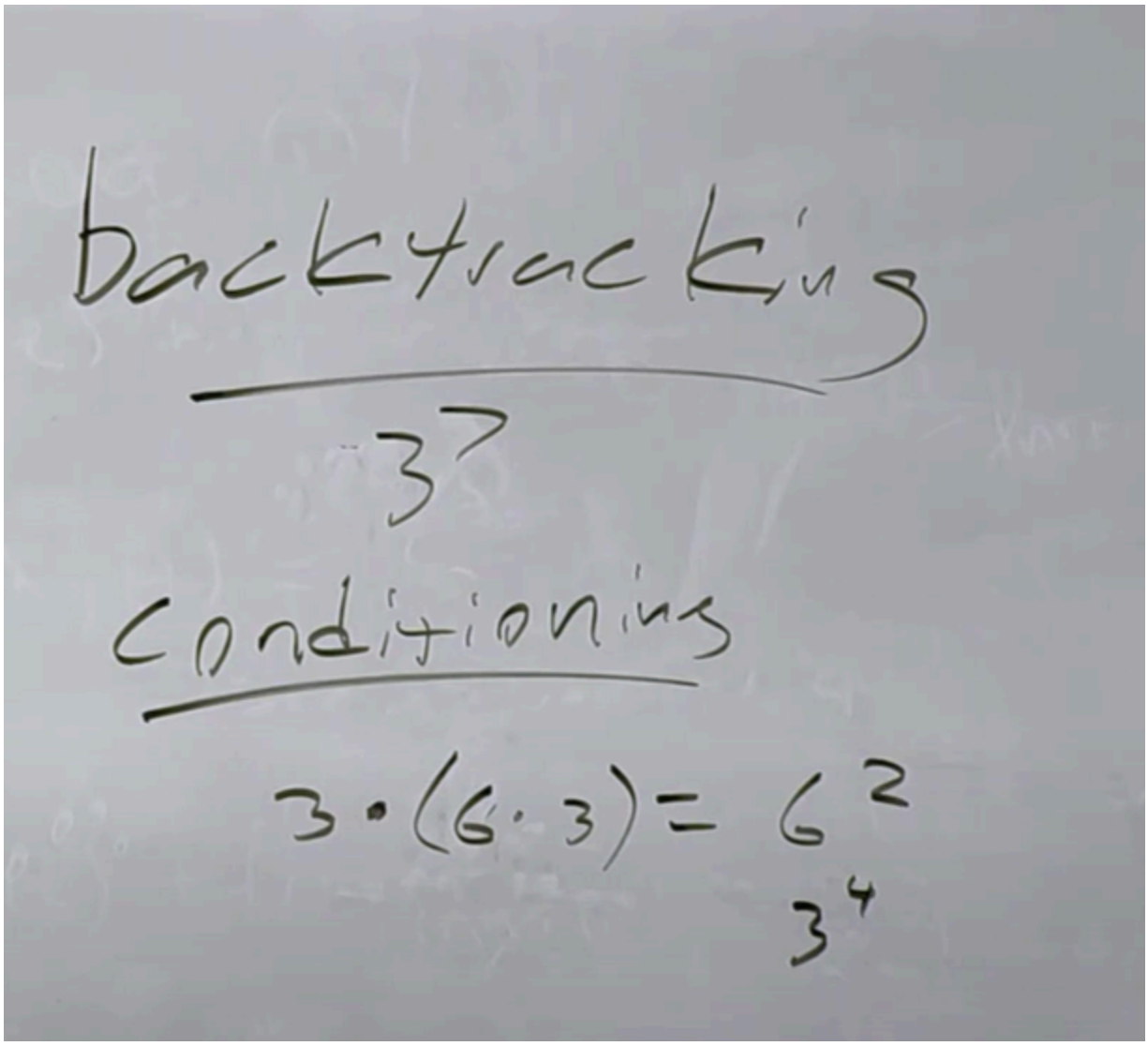
So we can just repeatedly condition the value and solve



So this becomes very quick and you can read the maximum weight very easily

> This covers all the possible combinations that backtracking could have
> But this is much faster, cuz we are taking a very complicated problem and break down into easily solvable pieces

In this example we are traking an exponential problem and break it into a linear number of linear solvable problems



- Now that we understand conditional independence, how is it useful?
- First, this formalizes the fact that if someone tells you the value of a variable, you can condition on that variable, thus potentially breaking down the problem into simpler pieces.
- If we are not told the value of a variable, we can simply try to condition on all possible values of that variable, and solve the remaining problem using any method. If conditioning breaks up the factor graph into small pieces, then solving the problem becomes easier.
- In this example, conditioning on $X_1 = v$ results in a fully disconnected graph, the maximum weight assignment for which can be computed in time linear in the number of variables.

Summary so far

Independence: when sets of variables A and B are disconnected; can solve separately.

Conditioning: assign variable to value, replaces binary factors with unary factors

Conditional independence: when C blocks paths between A and B

Markov blanket: what to condition on to make A conditionally independent of the rest.

Elimination