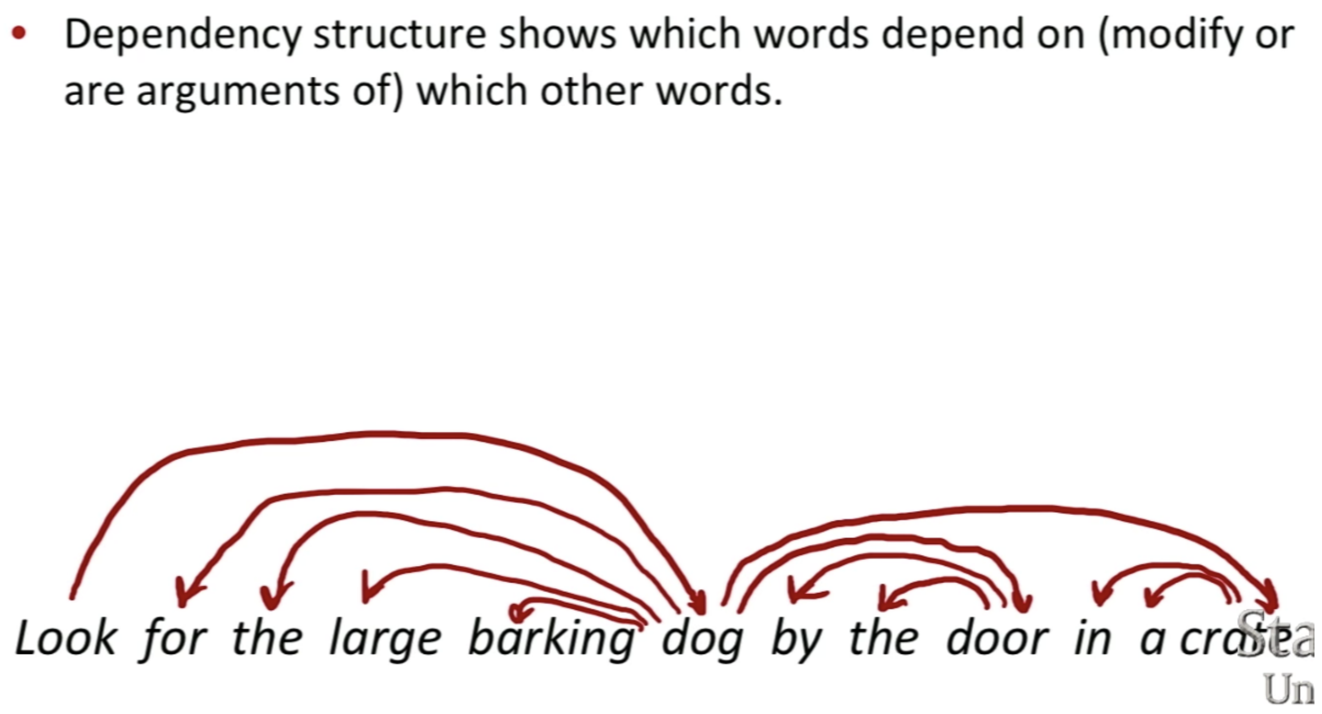
# dependency parsing and POS tagging

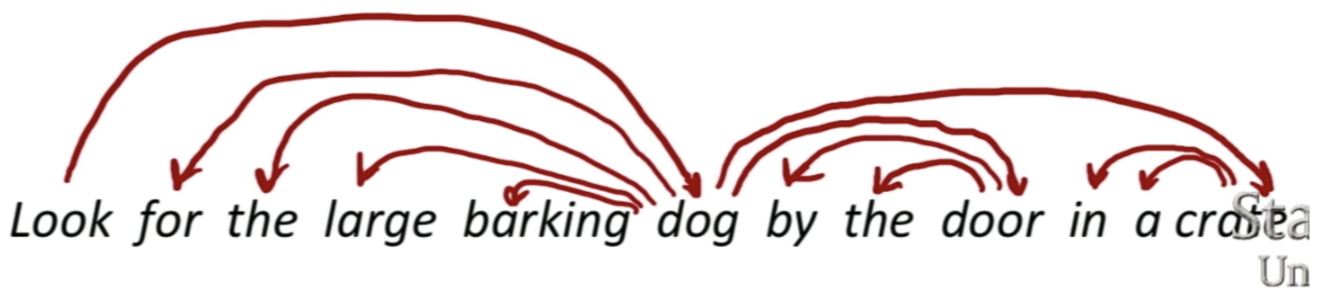
- a **treebank** is a parsed text corpus that annotates syntactic or semantic sentence structure.
- in the past Noam Chomsky and others have come up with formal rules for grammar and how to decide what is legal vs not.
- another way to look at it is instead of having formal rules and context free grammars, we can look at dependency parsing. this means figuring out for each word, which word that word modifies, and constructing a tree out of such dependencies. this approach, vs the formal rules approach, has turned out to work really well with deep learning in the last few years.
- treebanks give us the labelled datasets of such dependency parsing



- Dependency structure shows which words depend on (modify or are arguments of) which other words.

- this kind of structure and figuring out what is modifying what can be ambiguous and upto interpretation based on context and semantics. examples of confusion:

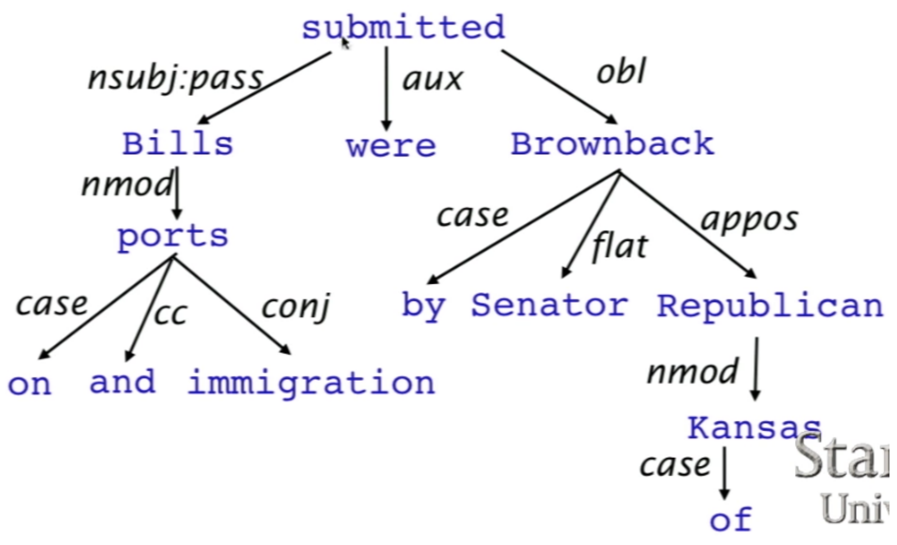
- Dependency structure shows which words depend on (modify or are arguments of) which other words.



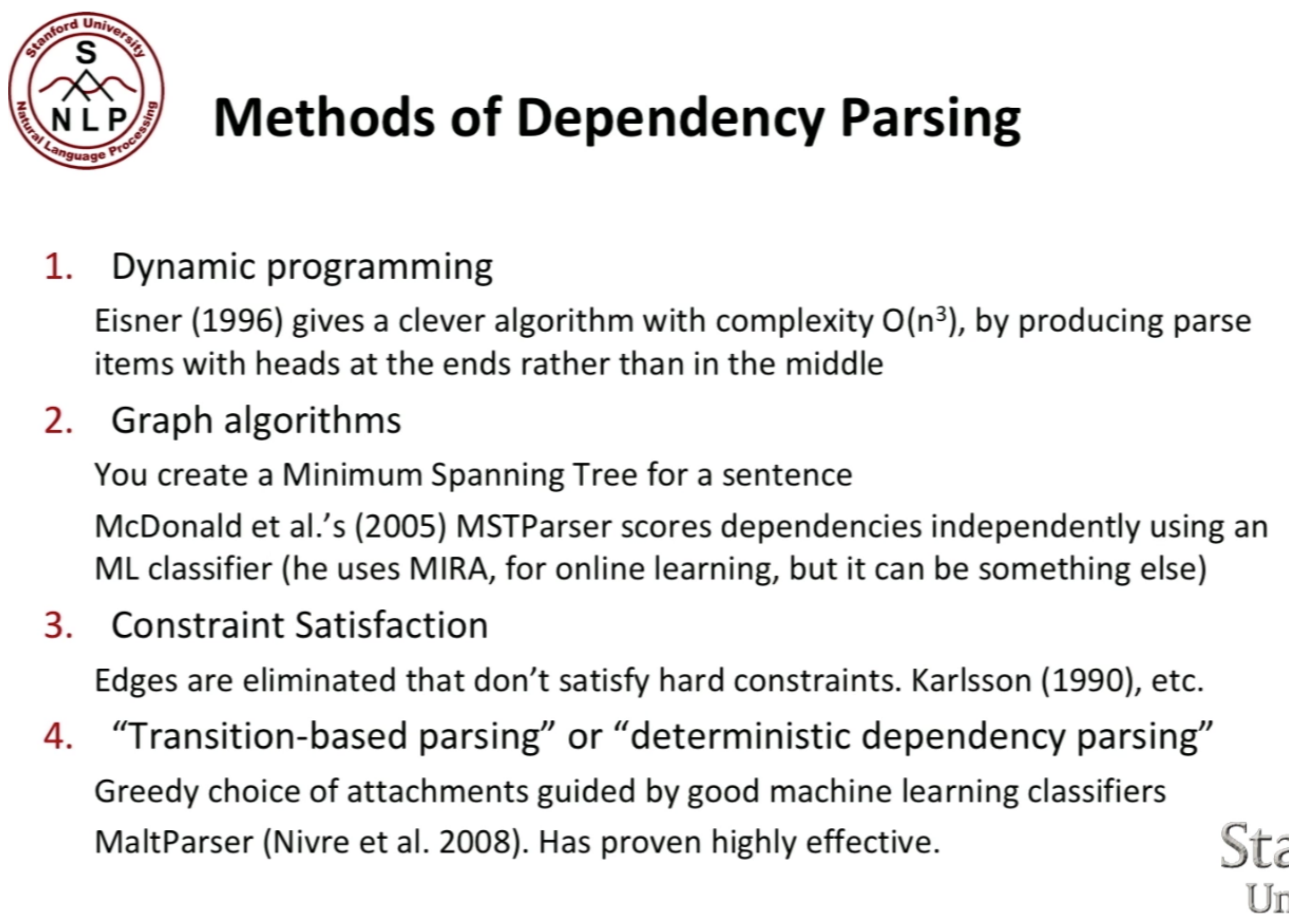
- also, there are typed versions of this dependency parsing where they attach types to each arrow:

The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



- when the sentence is written and the dependency arrows are drawn, if you can draw out the arrows so that no arrow crosses another, then it is called a projective dependency. if the arrows do cross, then its a non-projective dependency.
- dependency trees, defined  this way, you cannot recover the order of words from them, in general.
- methods of dependency parsing → we will focus on and build on the 4th one



## Methods of Dependency Parsing

1. Dynamic programming

   Eisner (1996) gives a clever algorithm with complexity $O(n^3)$, by producing parse items with heads at the ends rather than in the middle

2. Graph algorithms

   You create a Minimum Spanning Tree for a sentence

   McDonald et al.'s (2005) MSTParser scores dependencies independently using an ML classifier (he uses MIRA, for online learning, but it can be something else)

3. Constraint Satisfaction

   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

4. "Transition-based parsing" or "deterministic dependency parsing"

   Greedy choice of attachments guided by good machine learning classifiers

   MaltParser (Nivre et al. 2008). Has proven highly effective.

- the last ML based method is what is dominant nowadays in industry. it has worked really well and we will approach this with the tools of word vectors and classification with simple, feed forward neural nets.
- here, we will study arc based transition dependency parsers. here's the algorithm. there are 3 possible actions at each stage:
  - shift: move word at top of buffer (next word in sentence) to top of current stack
  - left-arc: make the second from top on stack dependent on the top of stack. remove the second from top from stack.
  - right-arc: make the top of stack dependent on the second top of stack. remove the top from stack.
- given the above three operations, the goal is to build a supervised learning classifier to decide what step to take at each stage. initially this was done with SVMs or logistic regression.
- another approach is to jointly learn both action as well as type of dependency to tag it with and then use this type of dependency as an input feature into the next step of parsing (next invocation of the classifier).
- the loss to train against could be unlabelled accuracy score (UAS ignores the POS tag) or the labelled accuracy score (with LAS label is only right if both arrow and label are right)
- doesn't this loss metric suffer from the fact that if we get something higher up in the tree wrong, we will penalize lower ones? yea, but usually not a problem since we can often get the downstream ones right even if we mis labelled or got the arrow wrong on an upstream one.
- Google's SyntaxNet, which is state of the art uses a variant of the above method.
- in the above methods of using SVMs or logistic regression, they still used sparse representation of words with dimension of vector proportional to vocab size. this sparsity made training and generalization very challening even with reasonably large datasets.

- enter neural nets with word embeddings (like word2vec or GloVe). more dense representations of words with much fewer dimensions. when we train a neural net with these word vectors as features for the dependency parsing task, we win. the accuracy is slightly better, but more importantly since we've reduced the dimensionality quite a bit, its a lot lot faster to train the dependency parsing model. (if you're goal is parse a large amount of text or parse the web, this is a huge scalability win)
- interestingly, they also learned embeddings for part of speech tags and dependency label types and used these as features for the dependency parsing neural net. they concatenate the vectors of words, POS and dependency types into one vector and feed that into the network. this is a common way to combine different things.