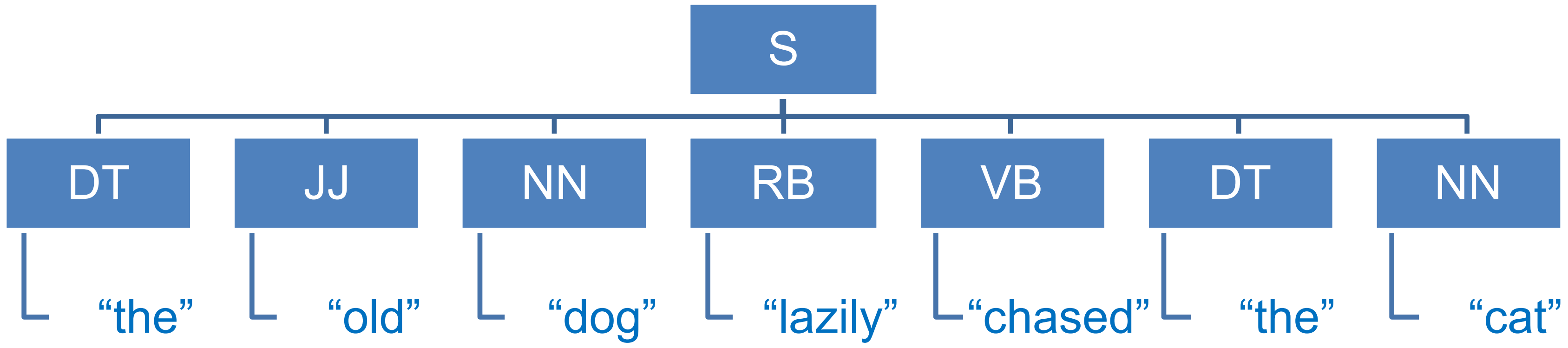
Syntactic Analysis: Shallow Parsing

Natural Language Processing

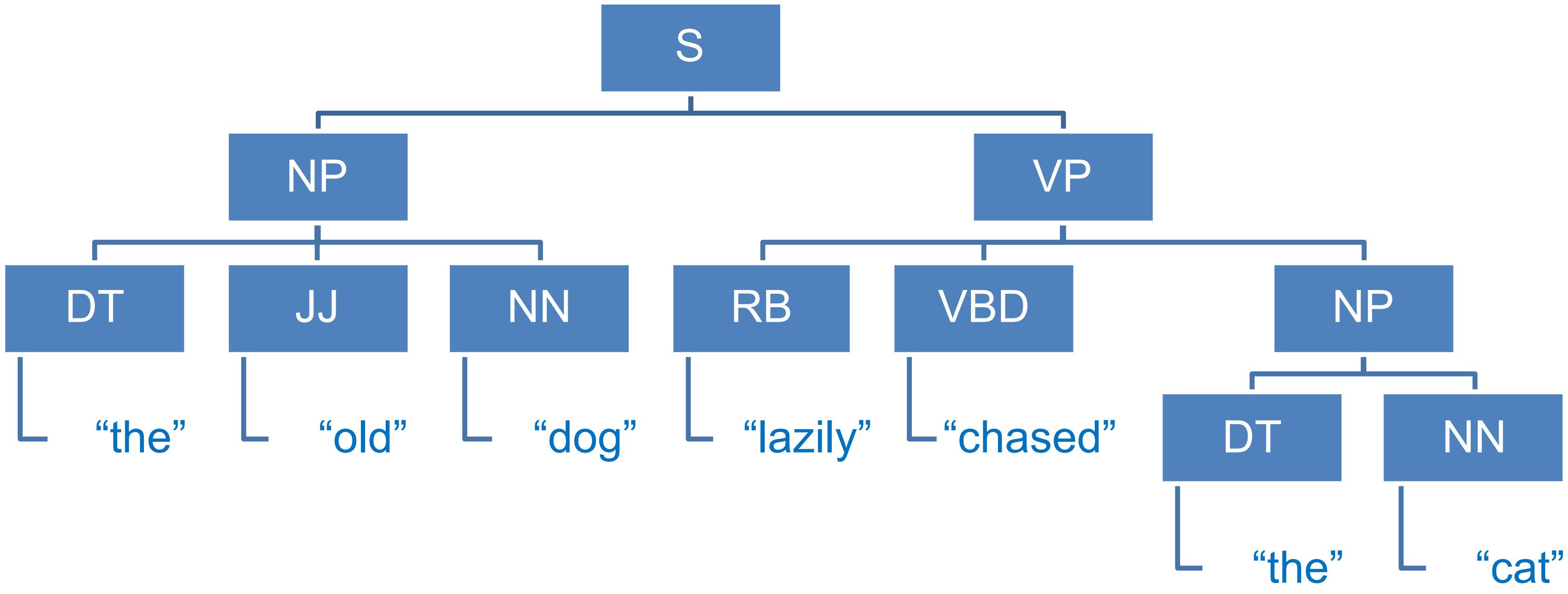
Shallow Parsing

- Shallowest syntactic parsing is POS tagging.
- Deepest is a full grammar tree.
- A middle level is “shallow parsing” or “chunking”—it groups POS tags into phrases but doesn’t decompose those phrases themselves.

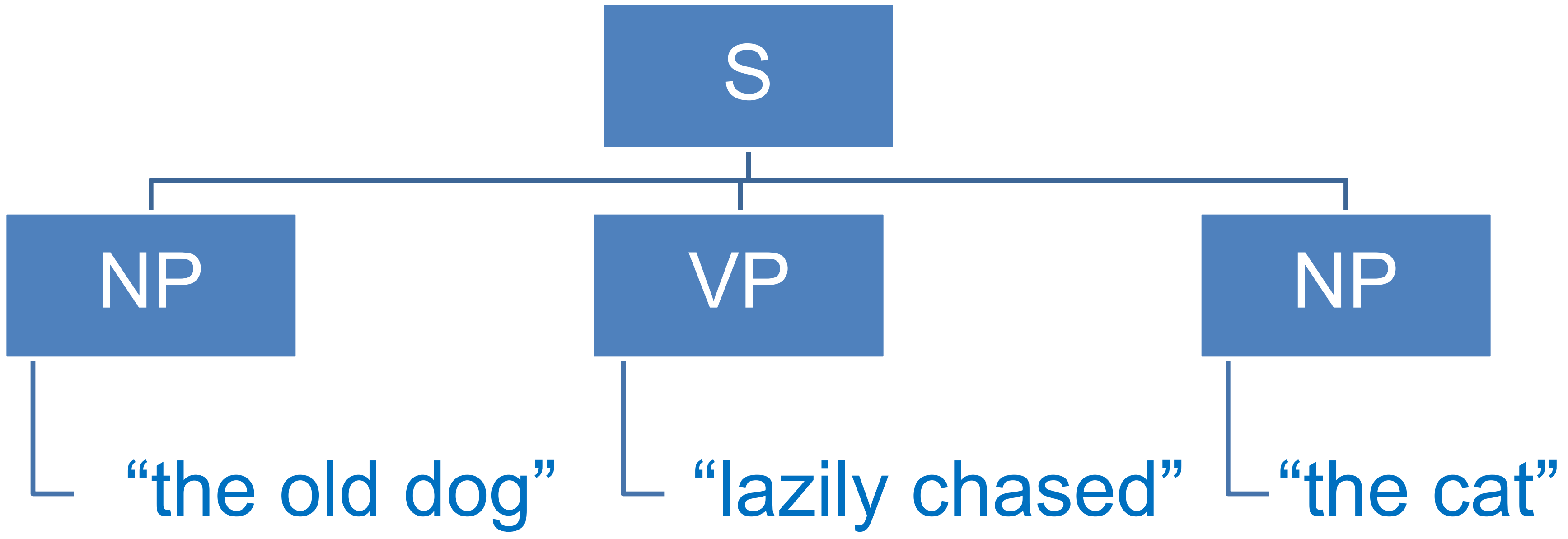
POS Only



Full Parse Tree

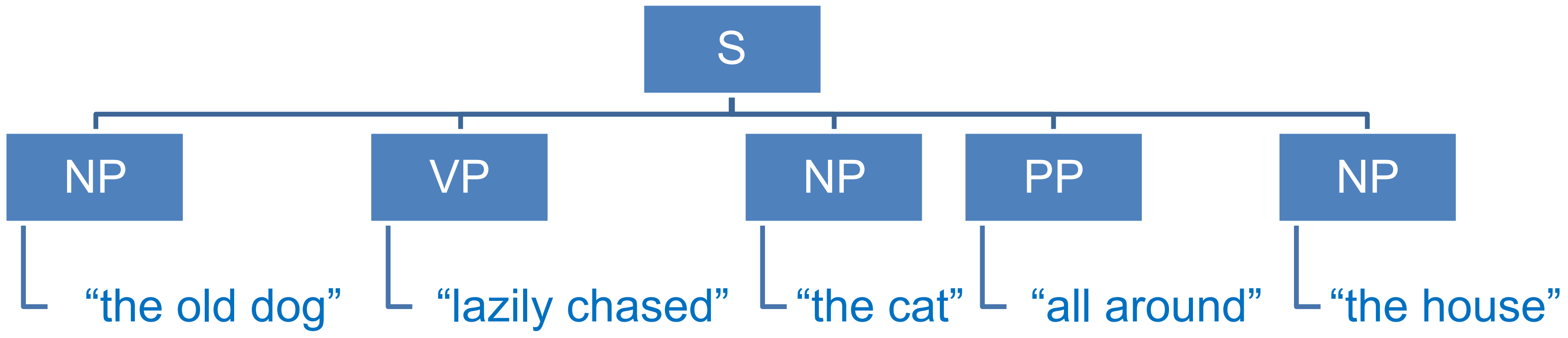


Shallow Parse Tree



Shallow Parse Tree

As the sentences get longer and longer, the chunker output remains easy on the eyes.

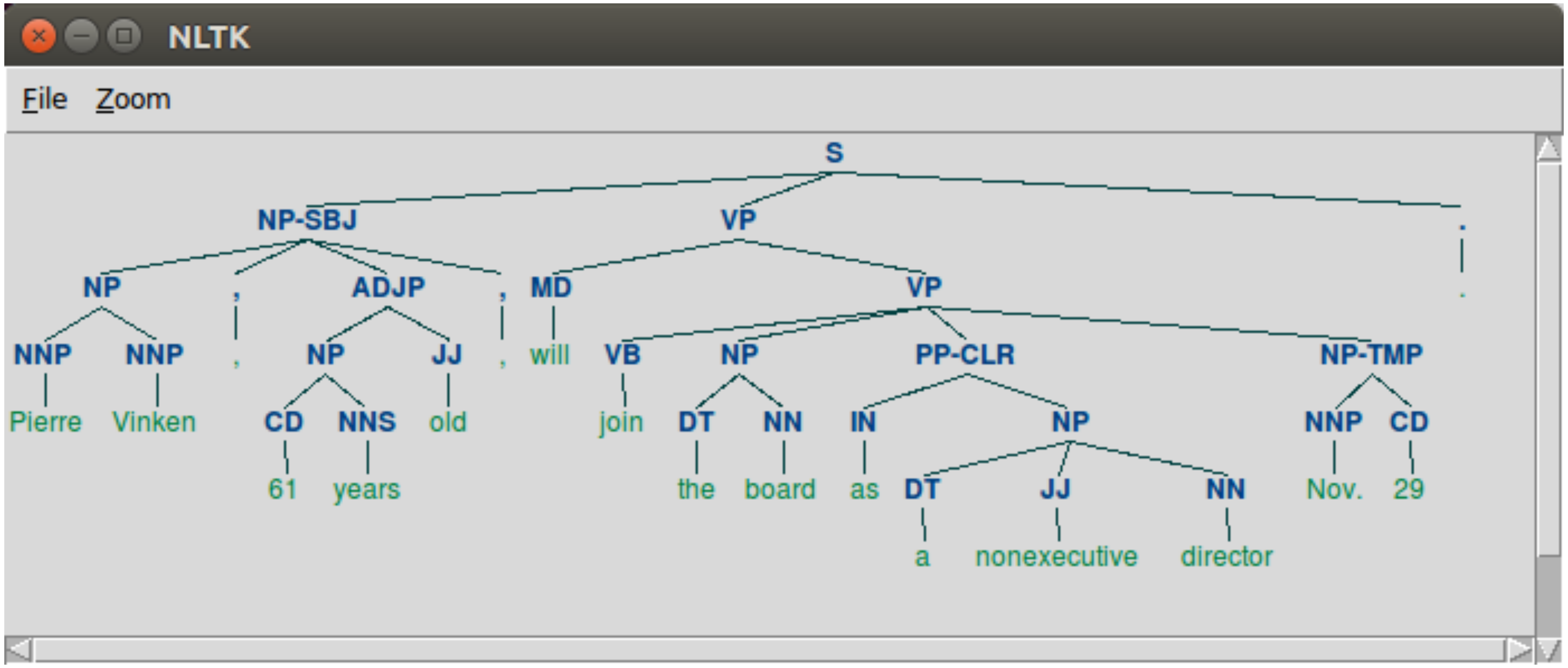


We call the tool that outputs this type of a tree a “chunker” because, as you can see, it breaks the sentence into useful, midsized chunks.

A typical chunk is much shorter than a sentence but usually more than just one word.

The Enormity of Full Parse Trees

That example was very simple. In real life, many full-depth parse trees look like this:



Why Chunk?

We might use chunking instead of full parsing because full parsing is:

- Slower (computationally expensive)
- Not super accurate on “grammatically challenged” text types, e.g., UGC*
- “Overkill” for many applications
  (If you’re not doing machine translation or question answering, then do you really need to break apart every subphrase?)

*user-generated content

Why Chunk?

We might use chunking instead of just POS tagging because POS tagging is:

- Not smart enough—doesn’t group words into noun phrases, verb phrases
- A short step from chunking—puts us at a point where there’s very little extra cost to add the chunking layer

DataScience@SMU

Syntactic Analysis:
Shallow Parsing—How to Chunk

Natural Language Processing

So How Do We Chunk?

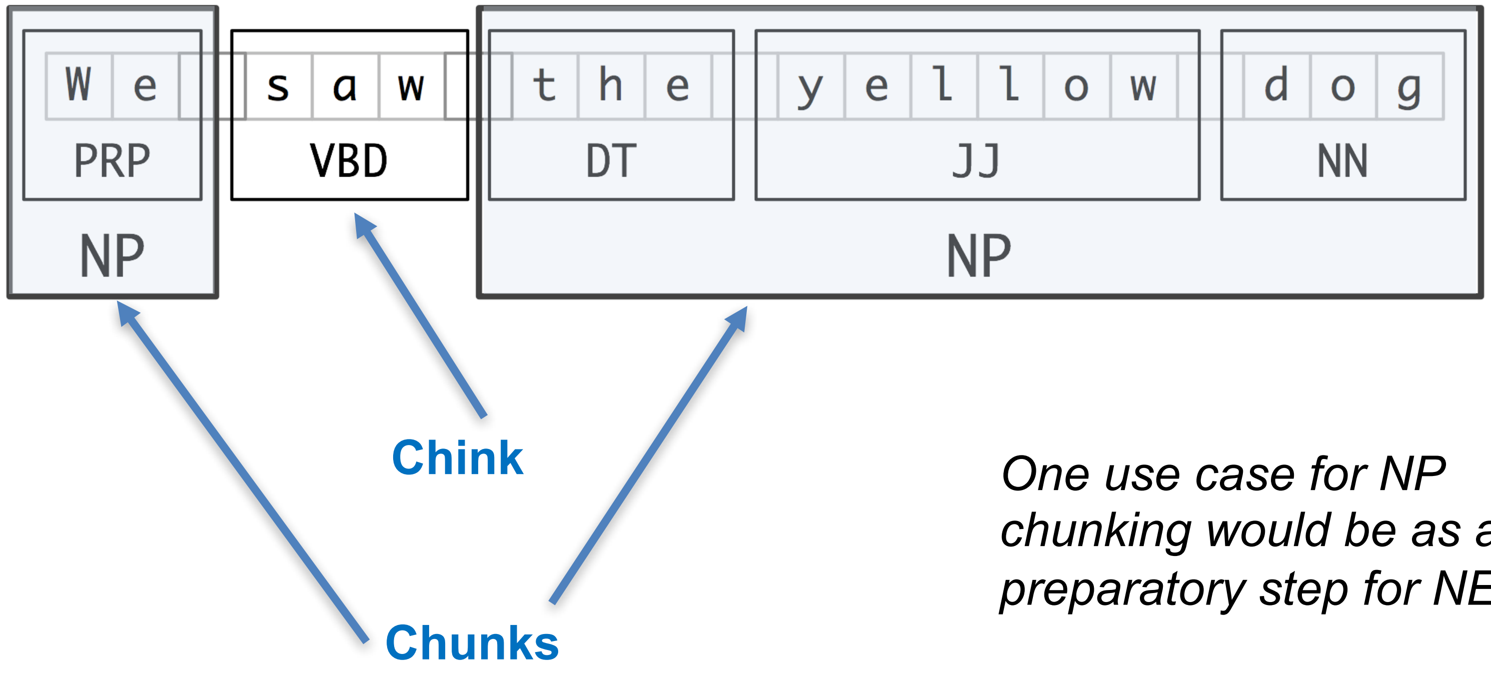
Well, the most straightforward way would be to RegEx for some obvious tag patterns.

```
NP: {<DT>?<JJ>*<NN>} # chunking patterns
VP: {<VBD>?<TO>?<VB>?}
```

This is a fast and easy way to get a basic chunker up and running.

NP Chunking, and Chinking

- In many applications, we are interested only in one type of phrase, e.g., noun phrases (NPs) and not VPs or PPs.
- Whatever syntactic elements we leave out of our chunks are called “chinks.”



One use case for NP chunking would be as a preparatory step for NER.

So How Do We Chunk?

Now we need to think about adding chinking patterns to our RegEx patterns.

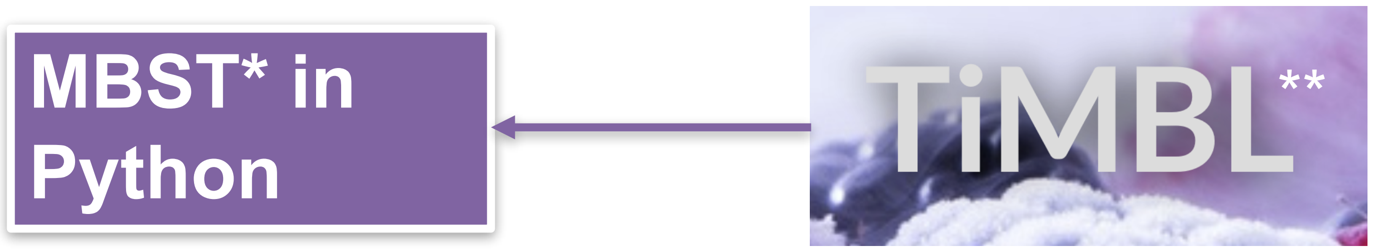
```
NP:  {<DT>?<JJ>*<NN>} # chunking patterns
VP:  {<VBD>?<TO>?<VB>?}
NP:  }<IN|DT>+{  # chinking pattern
```

Clearly we would need ***many*** more RegEx patterns to handle all the verbiage in a real text corpus.

So this approach is difficult to maintain and scale.

Classifier-Based Chunkers

- Fortunately, NLTK and other NLP toolkits offer a classifier-based chunker, so you don’t have to think up myriad RegEx patterns. Whew!
- For example, the pattern.en (or pattern3.en) package for Python has a shallow parser built with TiMBL (a decision-tree based learning classifier).



**memory-based shallow tagger*

***Tilburg Memory-Based Learner*

Annotation for Chunks

There is a standard annotation called “IOB”:

I	a token is inside a chunk
O	a token is outside any chunk (it’s a chink)
B	a token begins a new chunk

Annotation for Chunks

"**Most folks** inside of Boston city limits are pretty big New England Patriots fans."

("Most/JJS/**B-NP**/O folks/NNS/**I-NP**/O inside/IN/**B-PP**/B-PNP of/IN/**I-PP**/I-PNP Boston/NNP/**B-NP**/I-PNP city/NN/**I-NP**/I-PNP limits/NNS/**I-NP**/I-PNP are/VBP/**B-VP**/O pretty/RB/**B-NP**/O big/JJ/**I-NP**/O New/NNP/**I-NP**/O England/NNP-LOC/**I-NP**/O Patriots/NNPS/**I-NP**/O fans/NNS/**I-NP**/O")]

Annotation for Chunks

"Most folks **inside of** Boston city limits are pretty big New England Patriots fans."

("Most/JJS/**B-NP**/O folks/NNS/**I-NP**/O inside/IN/**B-PP**/B-PNP of/IN/**I-PP**/I-PNP Boston/NNP/**B-NP**/I-PNP city/NN/**I-NP**/I-PNP limits/NNS/**I-NP**/I-PNP are/VBP/**B-VP**/O pretty/RB/**B-NP**/O big/JJ/**I-NP**/O New/NNP/**I-NP**/O England/NNP-LOC/**I-NP**/O Patriots/NNPS/**I-NP**/O fans/NNS/**I-NP**/O")]

Annotation for Chunks

"Most folks inside of **Boston city limits** are pretty big New England Patriots fans."

("Most/JJS/**B-NP**/O folks/NNS/**I-NP**/O inside/IN/**B-PP**/B-PNP of/IN/**I-PP**/I-PNP Boston/NNP/**B-NP**/I-PNP city/NN/**I-NP**/I-PNP limits/NNS/**I-NP**/I-PNP are/VBP/**B-VP**/O pretty/RB/**B-NP**/O big/JJ/**I-NP**/O New/NNP/**I-NP**/O England/NNP-LOC/**I-NP**/O Patriots/NNPS/**I-NP**/O fans/NNS/**I-NP**/O")]

Annotation for Chunks

"Most folks inside of Boston city limits **are** pretty big New England Patriots fans."

("Most/JJS/**B-NP**/O folks/NNS/**I-NP**/O inside/IN/**B-PP**/B-PNP of/IN/**I-PP**/I-PNP Boston/NNP/**B-NP**/I-PNP city/NN/**I-NP**/I-PNP limits/NNS/**I-NP**/I-PNP are/VBP/**B-VP**/O pretty/RB/**B-NP**/O big/JJ/**I-NP**/O New/NNP/**I-NP**/O England/NNP-LOC/**I-NP**/O Patriots/NNPS/**I-NP**/O fans/NNS/**I-NP**/O")]

Annotation for Chunks

"Most folks inside of Boston city limits are **pretty big New England Patriots fans.**"

("Most/JJS/**B-NP**/O folks/NNS/**I-NP**/O inside/IN/**B-PP**/B-PNP of/IN/**I-PP**/I-PNP Boston/NNP/**B-NP**/I-PNP city/NN/**I-NP**/I-PNP limits/NNS/**I-NP**/I-PNP are/VBP/**B-VP**/O pretty/RB/**B-NP**/O big/JJ/**I-NP**/O New/NNP/**I-NP**/O England/NNP-LOC/**I-NP**/O Patriots/NNPS/**I-NP**/O fans/NNS/**I-NP**/O")]

Annotation for Chunks

("Most/JJS/**B-NP**/O folks/NNS/**I-NP**/O inside/IN/**B-PP**/B-PNP of/IN/**I-PP**/I-PNP Boston/NNP/**B-NP**/I-PNP city/NN/**I-NP**/I-PNP limits/NNS/**I-NP**/I-PNP are/VBP/**B-VP**/O pretty/RB/**B-NP**/O big/JJ/**I-NP**/O New/NNP/**I-NP**/O England/NNP-LOC/**I-NP**/O Patriots/NNPS/**I-NP**/O fans/NNS/**I-NP**/O")]

We don't have to assemble chunks from the IOB tags ourselves; we can call a function to do that.

Chunk('Most folks/NP'), Chunk('inside of/PP'), Chunk('Boston city limits/NP'), Chunk('are/VP'), Chunk('pretty big New England Patriots fans/NP')

DataScience@SMU

Syntactic Analysis: Working with Chunks

Natural Language Processing

Uses for Chunking

We've seen how we can create nice chunks:

```
Chunk('Most folks/NP'), Chunk('inside of/PP'),
Chunk('Boston city limits/NP'), Chunk('are/VP'),
Chunk('pretty big New England Patriots fans/NP')
```

Let's think about using this capability for NER ("named-entity recognition").

NER: Matching Registered Names

Why bother chunking if, for example, we just want to extract all the geographic place references in articles?

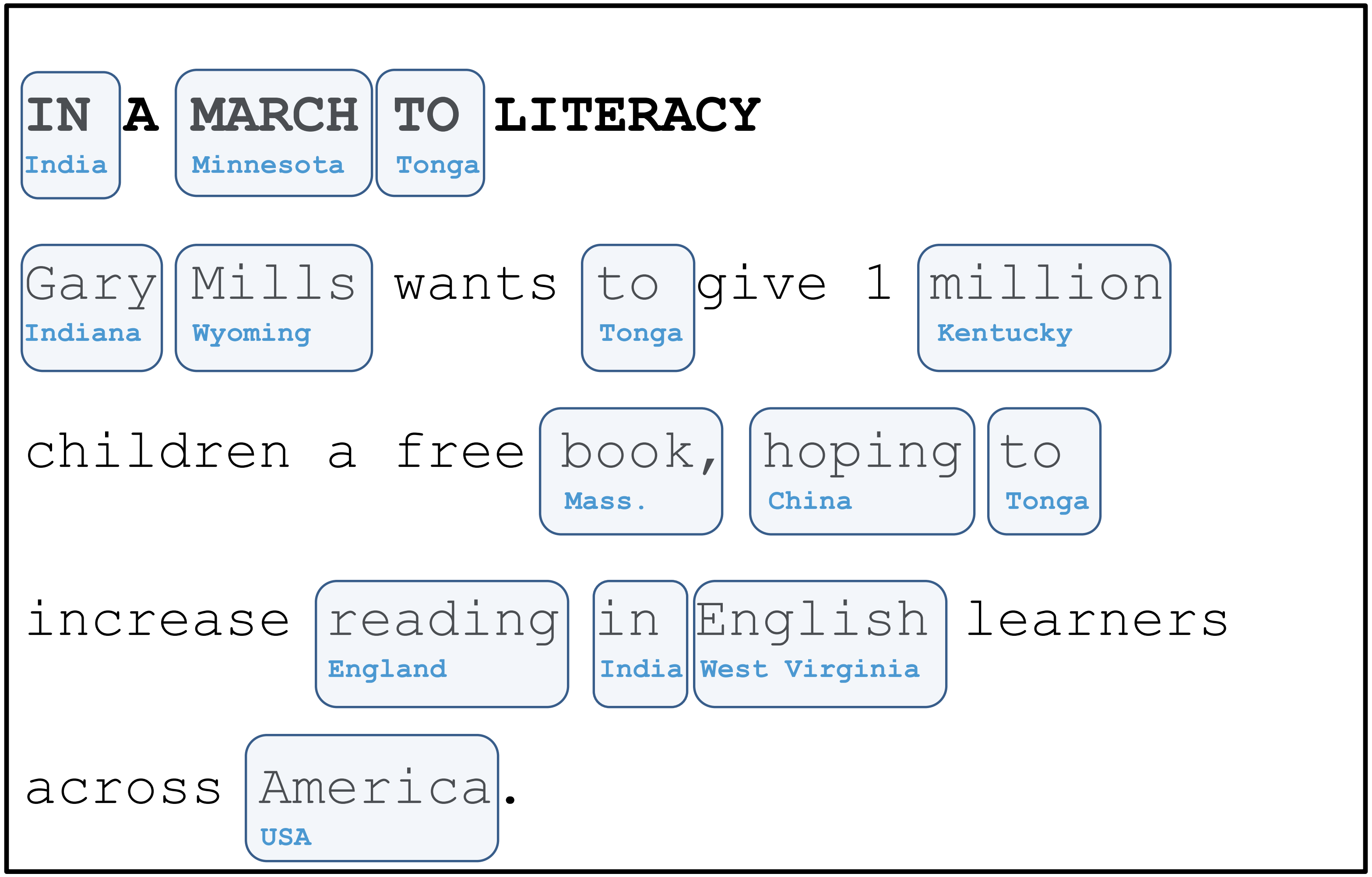
Can’t we just do look-ups from a registry of geographic names and abbreviations?

```
IN A MARCH TO LITERACY

Gary Mills wants to give 1 million
children a free book, hoping to
increase reading in English learners
across America.
```

NER: Matching Registered Names

Let's see how that might work…



IN A MARCH TO LITERACY

Gary Mills wants to give 1 million children a free book, hoping to increase reading in English learners across America.

NER: Matching Registered Names

NP chunking would narrow this down right away to this:

MARCH
LITERACY
Gary Mills
1 million children
free book
English learners
America

```
IN A MARCH TO LITERACY

Gary Mills wants to give 1
million children a free book,
hoping to increase reading in
English learners across
America.
```

NER: Matching Registered Names

We still have POS tags at our disposal, so let's strip out determiners, modifiers, and numbers.

MARCH
LITERACY
Gary Mills
~~1 million~~ children
~~free~~ book
~~English~~ learners
America

IN A MARCH TO LITERACY

Gary Mills wants to give 1 million children a free book, hoping to increase reading in English learners across America.

NER: Matching registered names

That leaves us with this:

MARCH
LITERACY
Gary Mills
children
book
learners
America

```
IN A MARCH TO LITERACY

Gary Mills wants to give 1
million children a free book,
hoping to increase reading in
English learners across
America.
```

NER: Matching Registered Names

Now let's look these up in our lexicon to see which are common nouns, and discard them.

~~MARCH~~
~~LITERACY~~
Gary Mills
~~children~~
~~book~~
~~learners~~
America

IN A MARCH TO LITERACY

Gary Mills wants to give 1 million children a free book, hoping to increase reading in English learners across America.

NER: Matching Registered Names

That leaves us with this:

```
IN A MARCH TO LITERACY

Gary Mills wants to give 1
million children a free book,
hoping to increase reading in
English learners across
America.
```

Gary Mills
America

NER: Matching Registered Names

Now we look these up in a registry of geographical names, for our last step of elimination.

~~Gary Mills~~
America

IN A MARCH TO LITERACY

Gary Mills wants to give 1 million children a free book, hoping to increase reading in English learners across America.

NER: Matching Registered Names

And that leaves us with this:

```
IN A MARCH TO LITERACY

Gary Mills wants to give 1
million children a free book,
hoping to increase reading in
English learners across
America.
```

America

The right result!

Note that this approach is still fallible, and an industrial-grade NER engine will have even more sophistication—but more on that later!

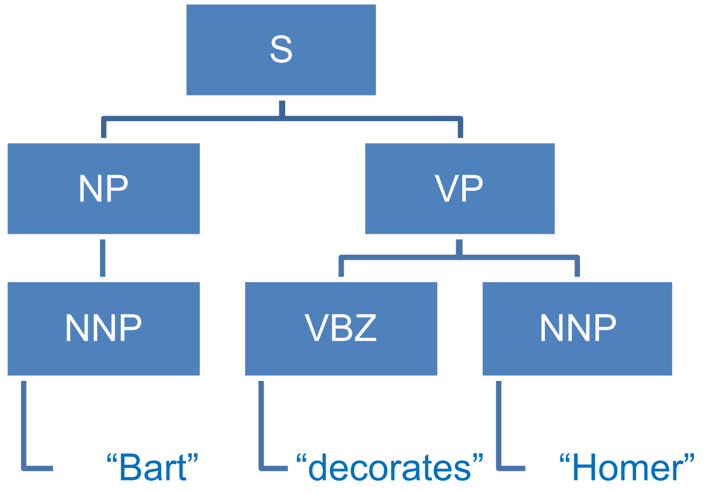
DataScience@SMU

Syntactic Analysis: Full Grammar Parsing

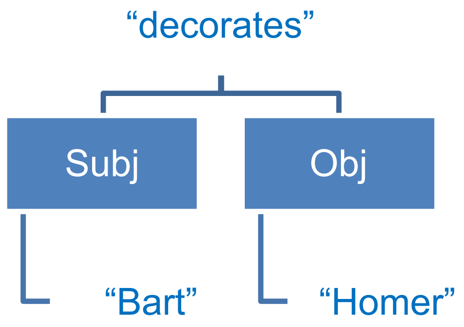
Natural Language Processing

Two Types of Parsers

Constituency Parse

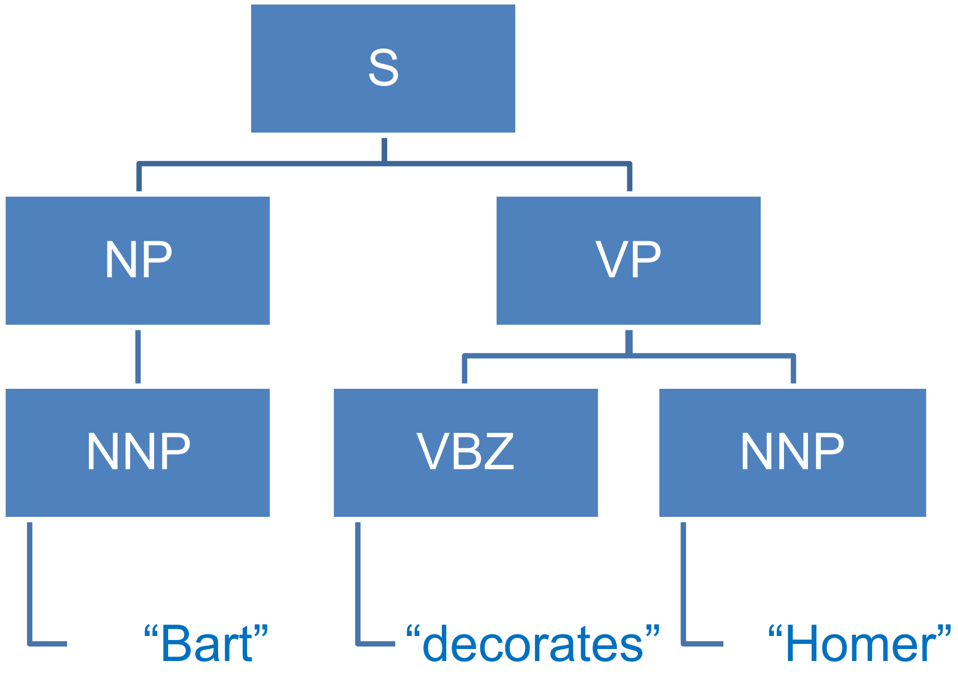


Dependency Parse



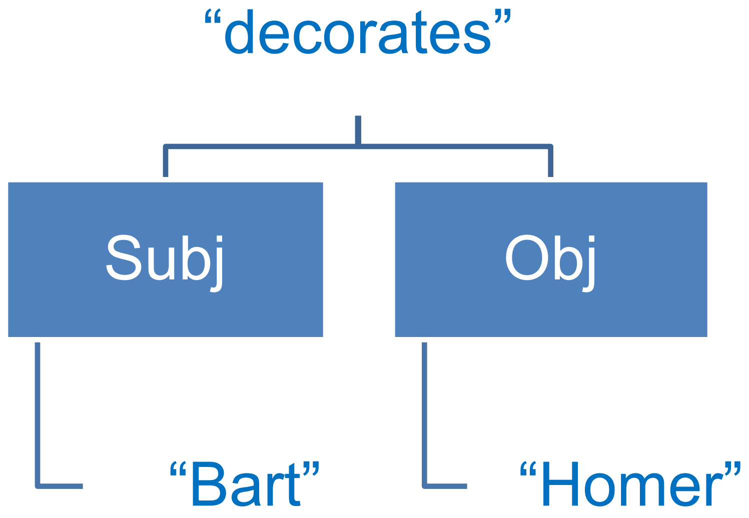
Two Types of Parsers

Constituency parsers break a sentence into sub-phrases and sub-sub-phrases, etc.



Two Types of Parsers

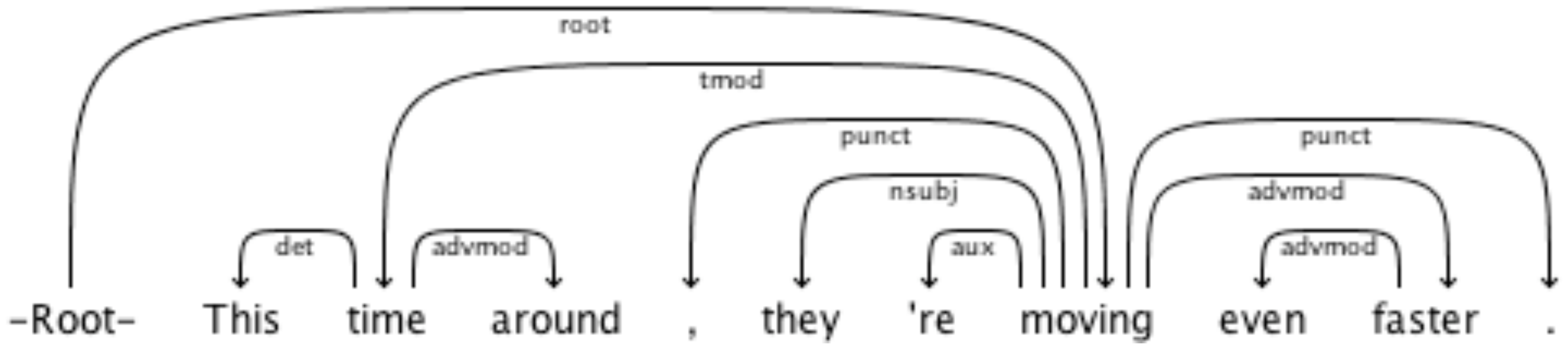
Dependency parsers give us labeled relations between words—this can give us, for example, the subject and object of the main verb.



Understanding Parse Trees

Features of a typical *dependency* parse graph:

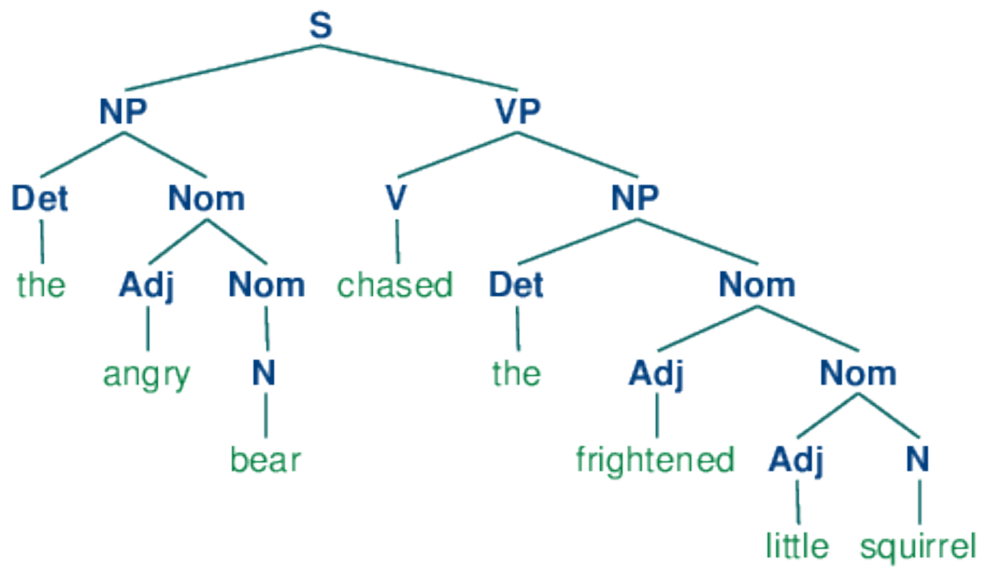
- Directed acyclic graph
- All nodes (beyond root) are words
- All edges are labeled
- Root node typically connects to the main verb



Understanding Parse Trees

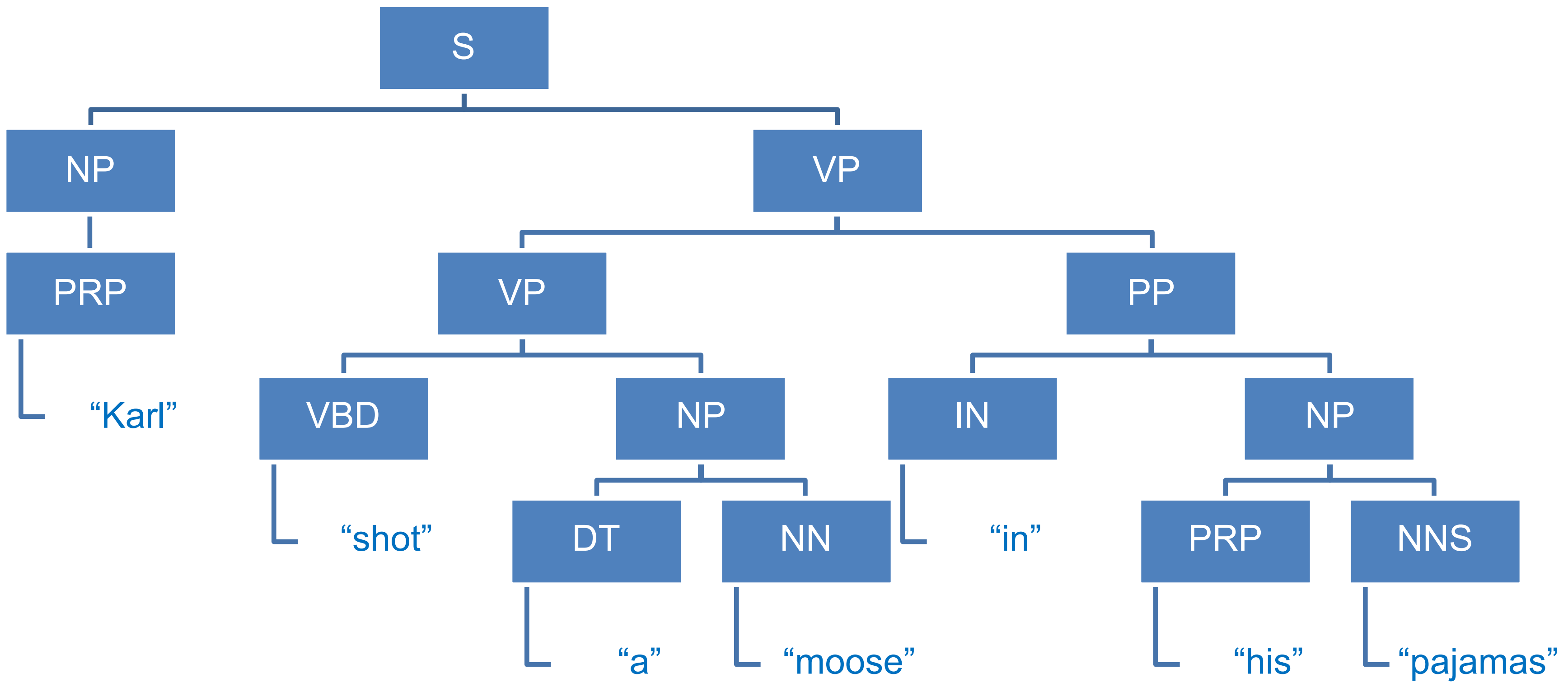
Features of a typical constituency parse graph:

- Directed acyclic graph
- Root node is the sentence
- Leaf nodes are words
- Penultimate nodes are POS tags
- All other interior nodes are phrase tags
- Edges are unlabeled



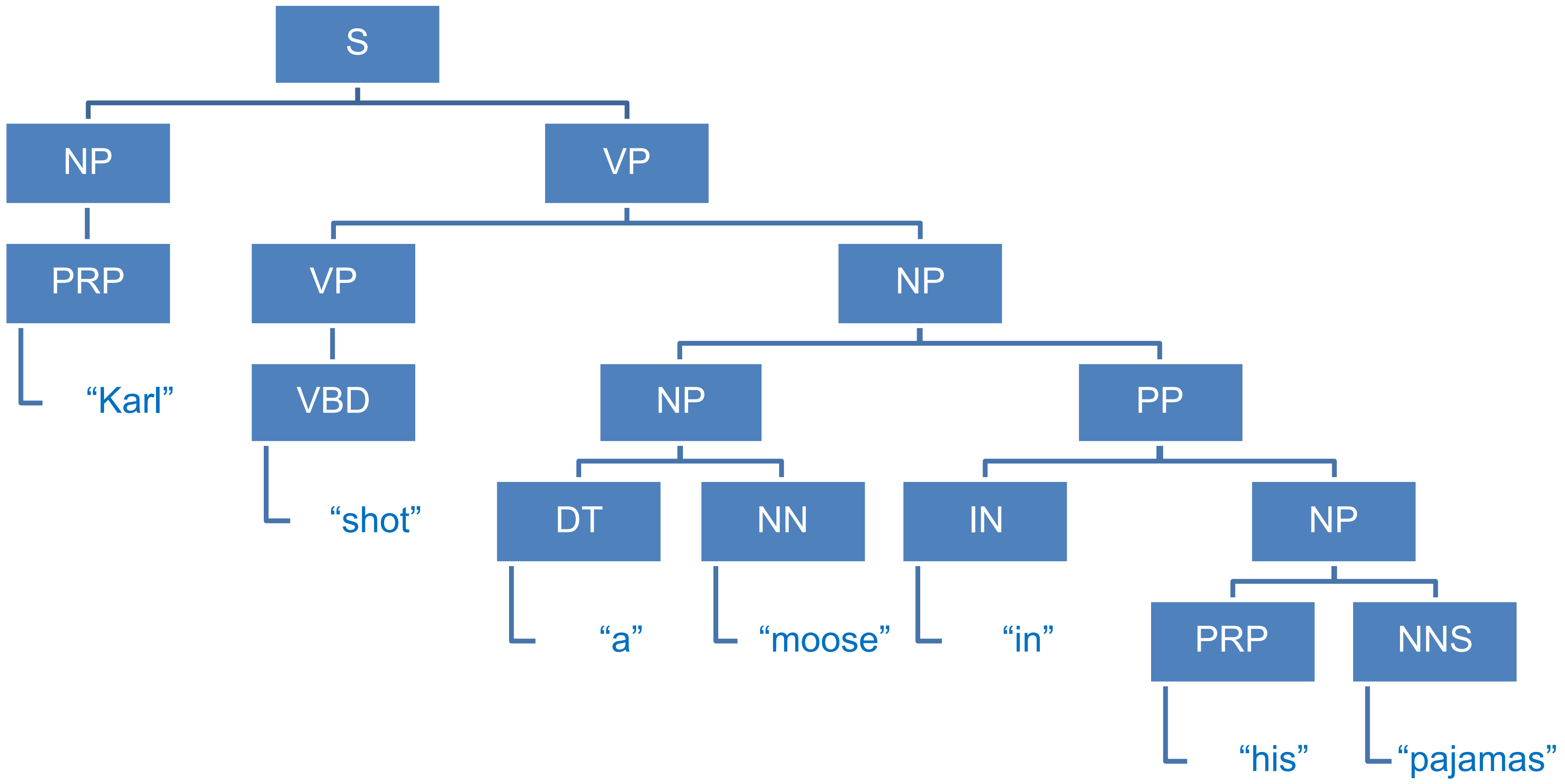
Ambiguity and Parse Trees

Who was wearing pajamas?



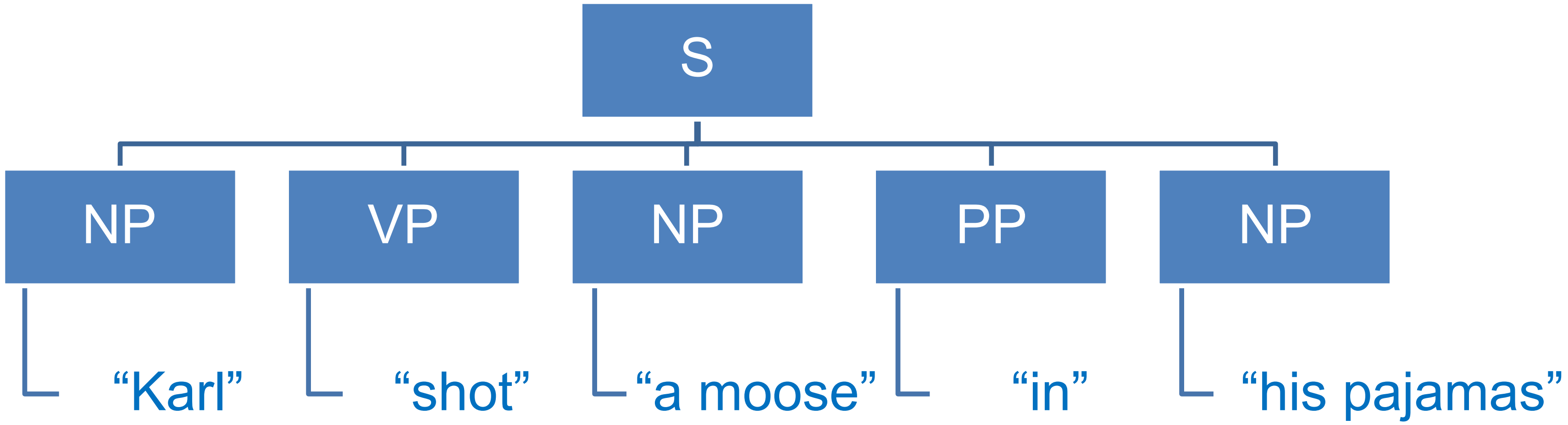
Ambiguity and Parse Trees

Who was wearing pajamas?



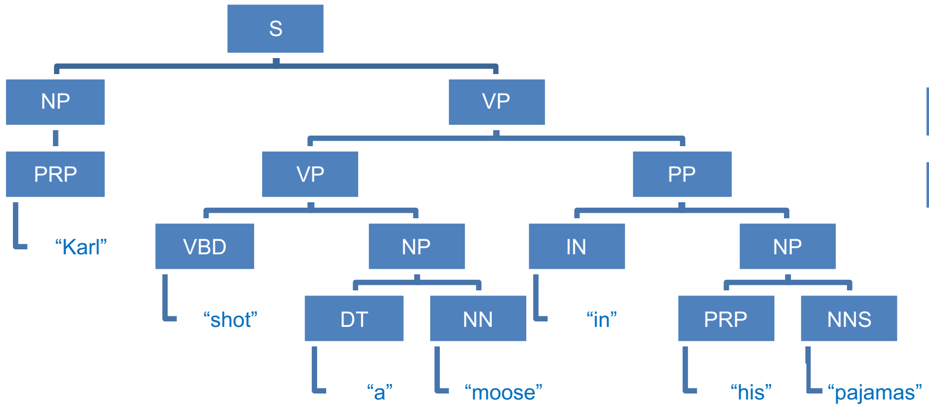
Ambiguity and Parse Trees

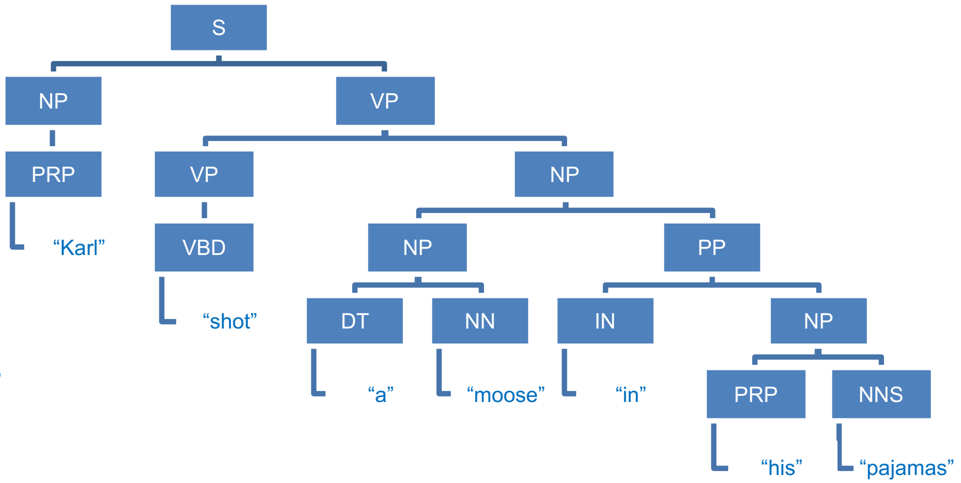
Notice the chunked version of both trees would be *the same.*



Ambiguity and Parse Trees

- This is a simple case—you can have longer sentences with even more alternative parse trees.
- A higher-level analysis involving semantics, pragmatics, or background knowledge would be needed to determine the "*right*" *(normative)* parse.





**probabilistic context-free grammar*

DataScience@SMU

Syntactic Analysis:

Full Grammar Parsing—Creating Parse Trees

Natural Language Processing

Creating Parse Trees

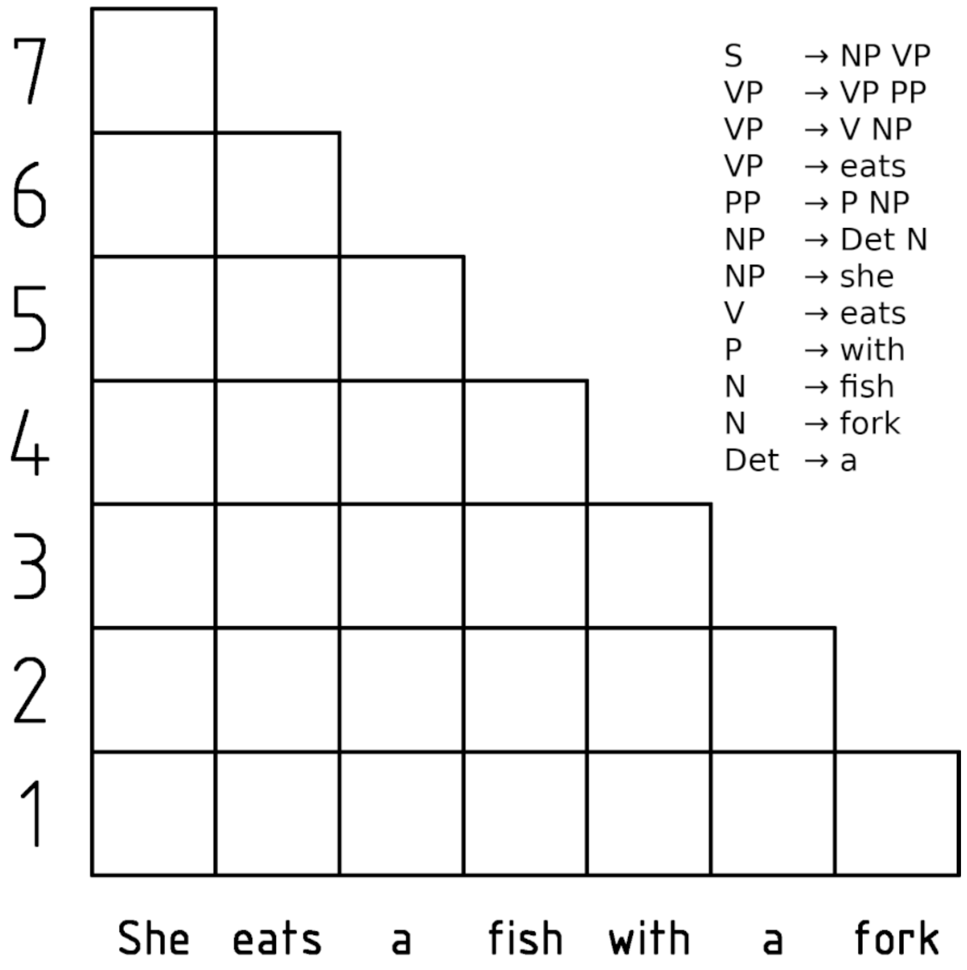
There are many different algorithms for creating parse trees, but perhaps the easiest to understand is the CYK* algorithm.

- It uses an increasingly large window to scan a sentence, labelling whether the whole window matches any existing grammar rule.
- Eventually the window widens to encompass the whole sentence and, hopefully, still matches the grammar!
- If so, we have a successful and complete parse of the sentence.

**Named after its inventors, John Cocke, Daniel Younger, and Tadao Kasami.*

CYK in Action

In this animation, notice how the procedure builds up levels of nested labels.

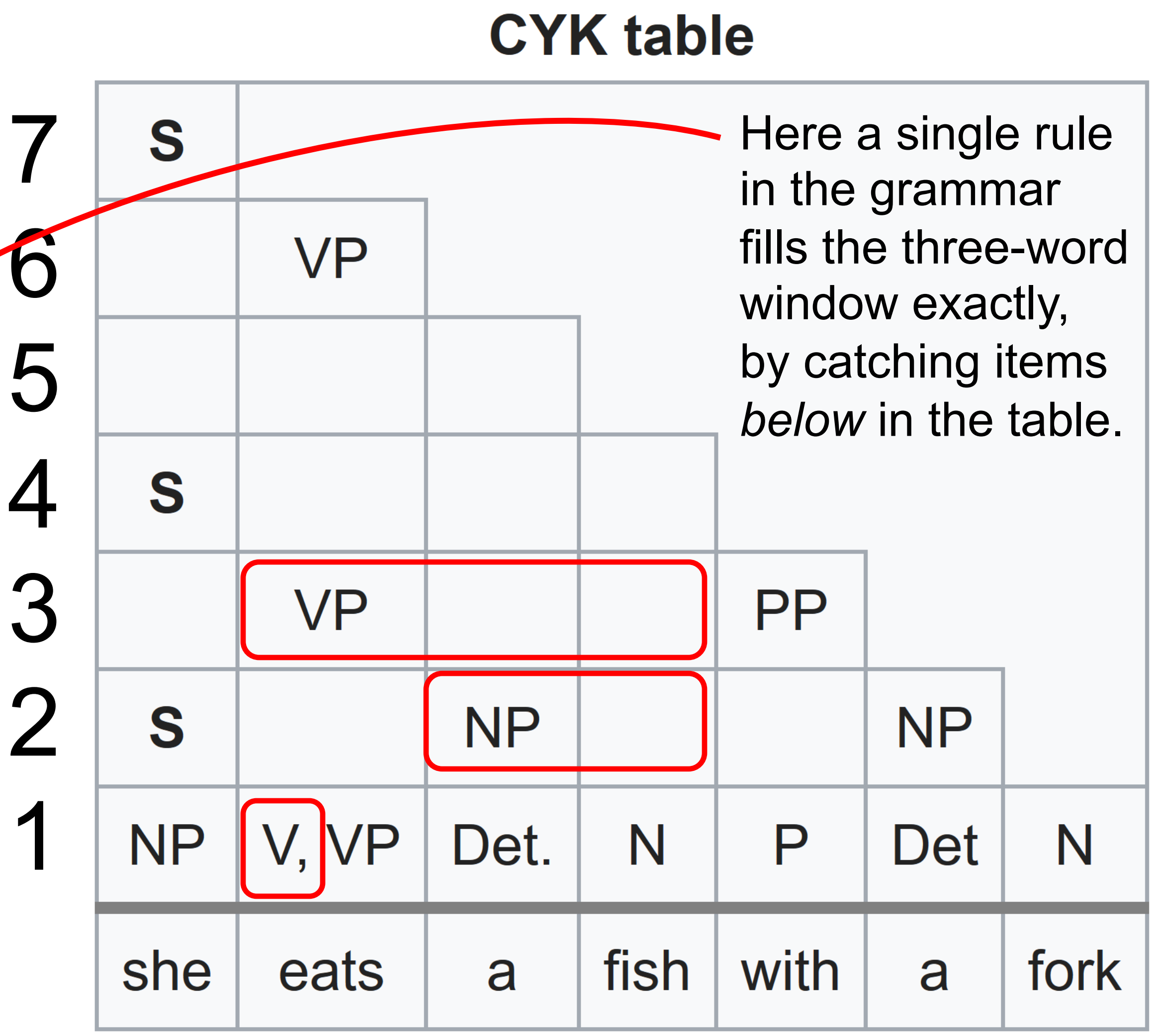


S	→ NP VP
VP	→ VP PP
VP	→ V NP
VP	→ eats
PP	→ P NP
NP	→ Det N
NP	→ she
V	→ eats
P	→ with
N	→ fish
N	→ fork
Det	→ a

CYK Detail

Focusing on a step…

$$S \longrightarrow NP\ VP$$
$$VP \longrightarrow VP\ PP$$
$$\boxed{VP \longrightarrow V\ NP}$$
$$VP \longrightarrow eats$$
$$PP \longrightarrow P\ NP$$
$$NP \longrightarrow Det\ N$$
$$NP \longrightarrow she$$
$$V \longrightarrow eats$$
$$P \longrightarrow with$$
$$N \longrightarrow fish$$
$$N \longrightarrow fork$$
$$Det \longrightarrow a$$



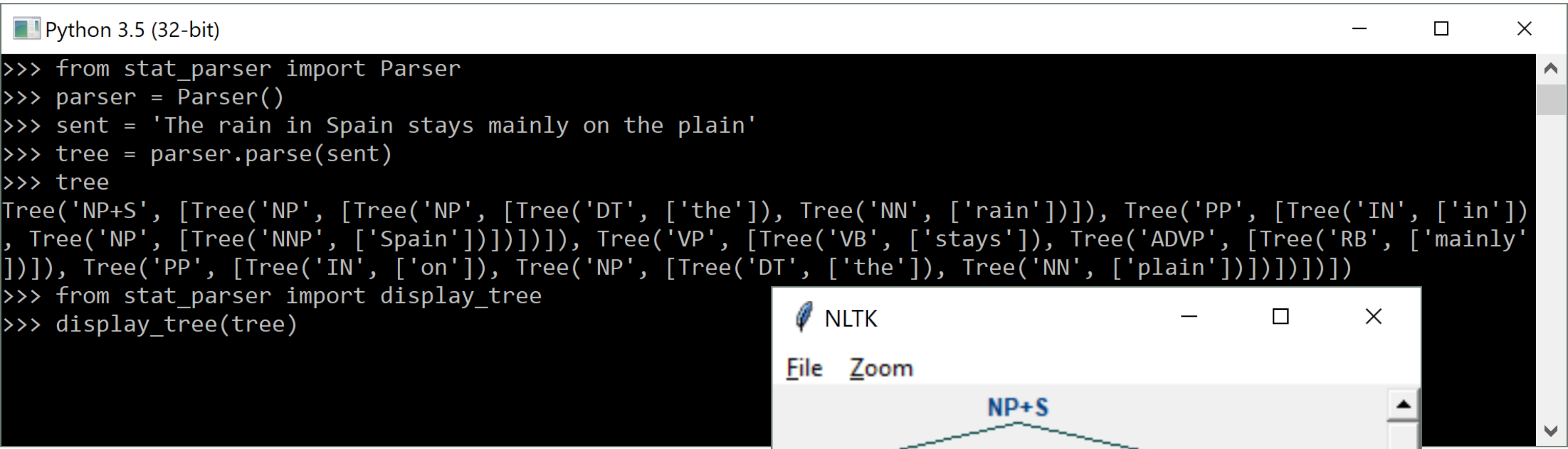
CYK table

7	S						
6		VP					
5							
4	S						
3		VP			PP		
2	S		NP			NP	
1	NP	V, VP	Det.	N	P	Det	N
	she	eats	a	fish	with	a	fork

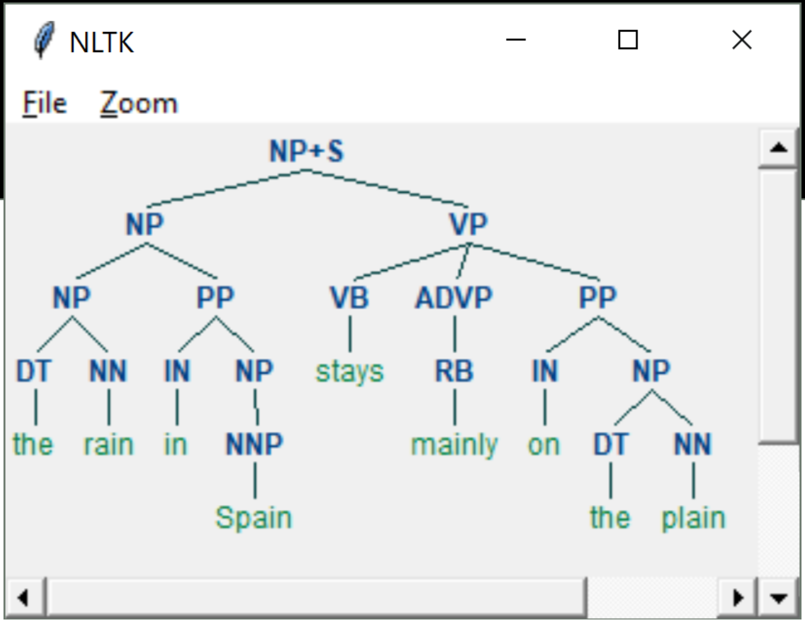
Here a single rule in the grammar fills the three-word window exactly, by catching items *below* in the table.

CYK for Everybody

It’s this simple to invoke CYK in Python:



```
>>> from stat_parser import Parser
>>> parser = Parser()
>>> sent = 'The rain in Spain stays mainly on the plain'
>>> tree = parser.parse(sent)
>>> tree
Tree('NP+S', [Tree('NP', [Tree('NP', [Tree('DT', ['the']), Tree('NN', ['rain'])]), Tree('PP', [Tree('IN', ['in'])
, Tree('NP', [Tree('NNP', ['Spain'])])])]), Tree('VP', [Tree('VB', ['stays']), Tree('ADVP', [Tree('RB', ['mainly'
])]), Tree('PP', [Tree('IN', ['on']), Tree('NP', [Tree('DT', ['the']), Tree('NN', ['plain'])])])])])
>>> from stat_parser import display_tree
>>> display_tree(tree)
```



- *This uses pyStatParser (available on GitHub), which extends CYK by adding weights to the grammar rules, i.e., it uses a PCFG.*
- *It can then take the highest-probability tree if it finds that more than one tree is possible.*

Other Parsers to Consider

There are plenty of parsers out there, and here are a few suggestions:

- Dependency parsing
  - MST parser
  - MALT parser
- Constituency parsing
  - Stanford parser
  - Link grammar parser

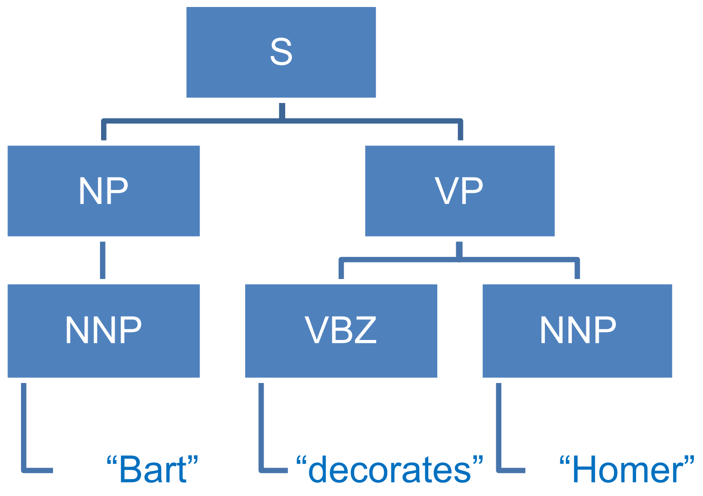
DataScience@SMU

Syntactic Analysis: Full Grammar Parsing

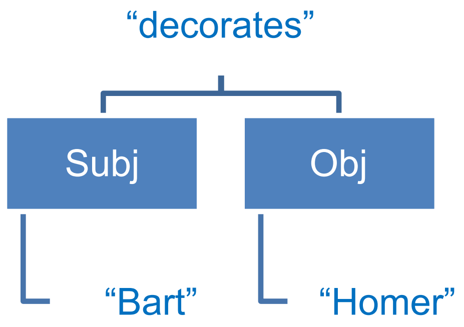
Natural Language Processing

Two Types of Parsers

Constituency Parse

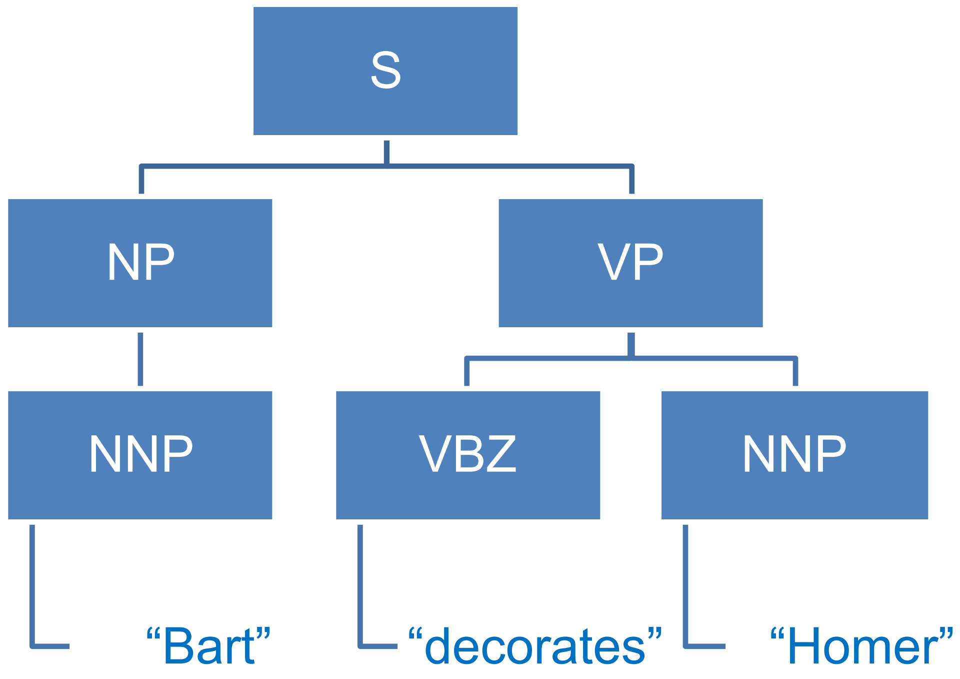


Dependency Parse



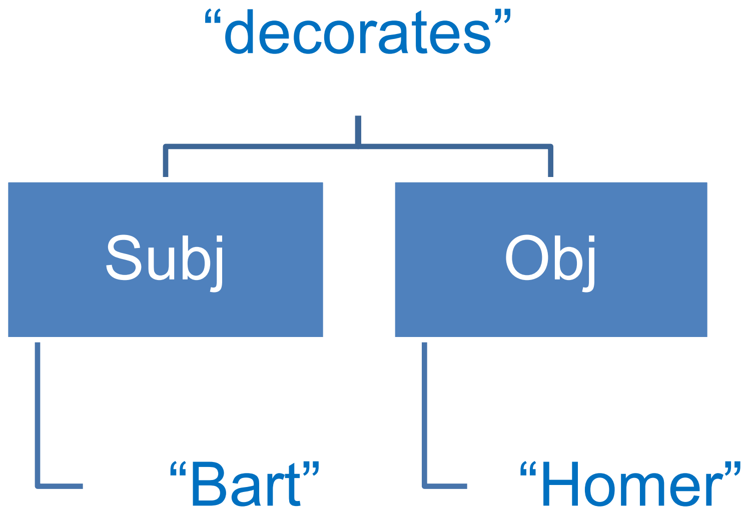
Two Types of Parsers

Constituency parsers break a sentence into sub-phrases and sub-sub-phrases, etc.



Two Types of Parsers

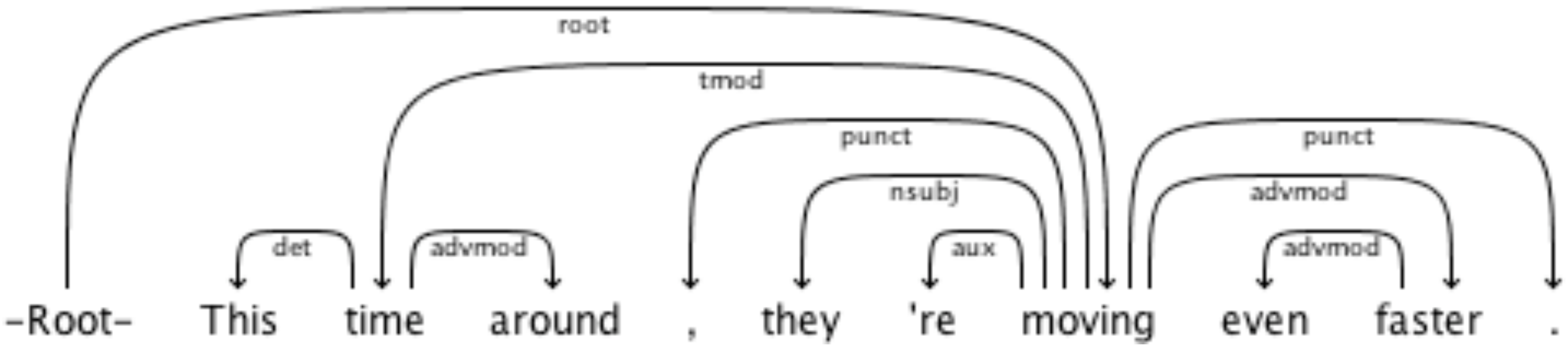
Dependency parsers give us labeled relations between words—this can give us, for example, the subject and object of the main verb.



Understanding Parse Trees

Features of a typical *dependency* parse graph:

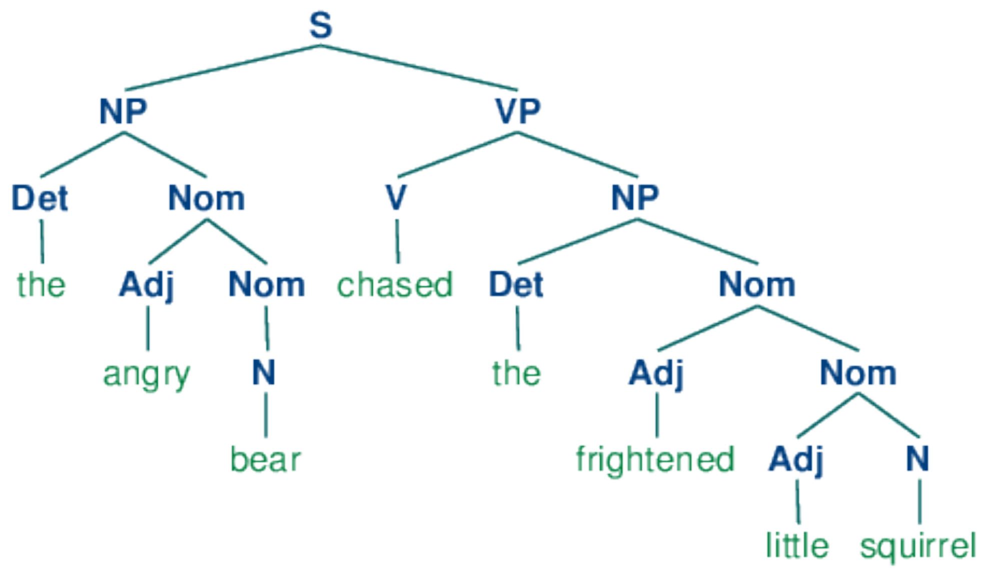
- Directed acyclic graph
- All nodes (beyond root) are words
- All edges are labeled
- Root node typically connects to the main verb



Understanding Parse Trees

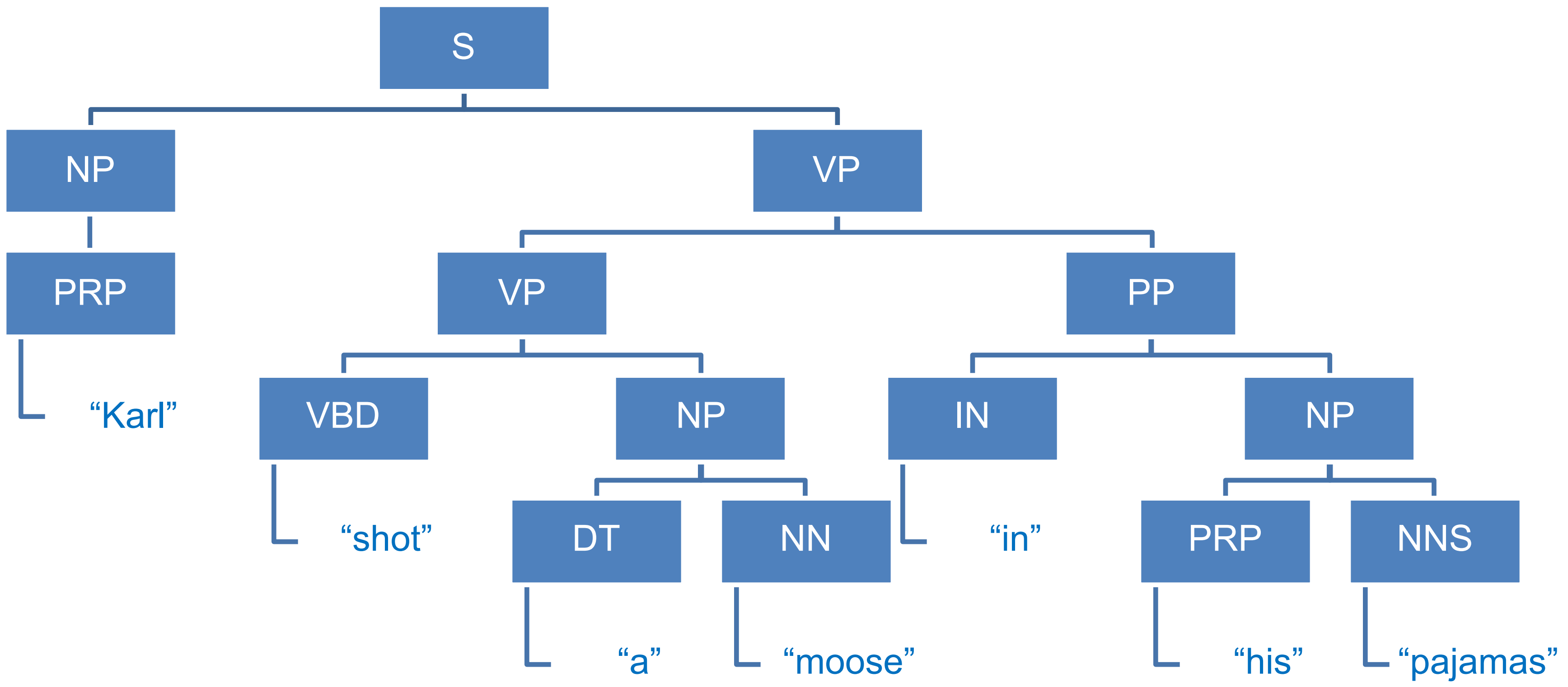
Features of a typical constituency parse graph:

- Directed acyclic graph
- Root node is the sentence
- Leaf nodes are words
- Penultimate nodes are POS tags
- All other interior nodes are phrase tags
- Edges are unlabeled



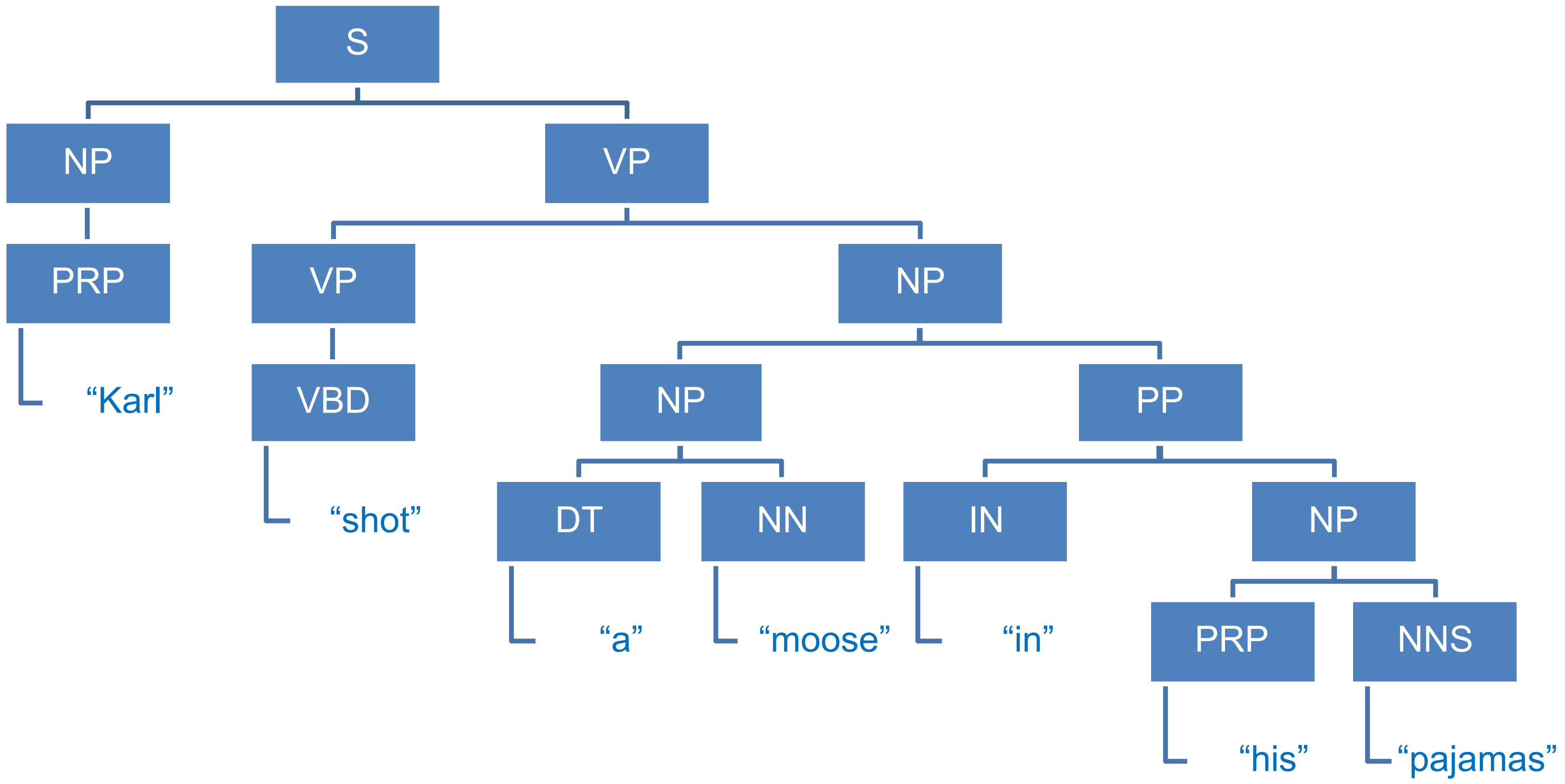
Ambiguity and Parse Trees

Who was wearing pajamas?



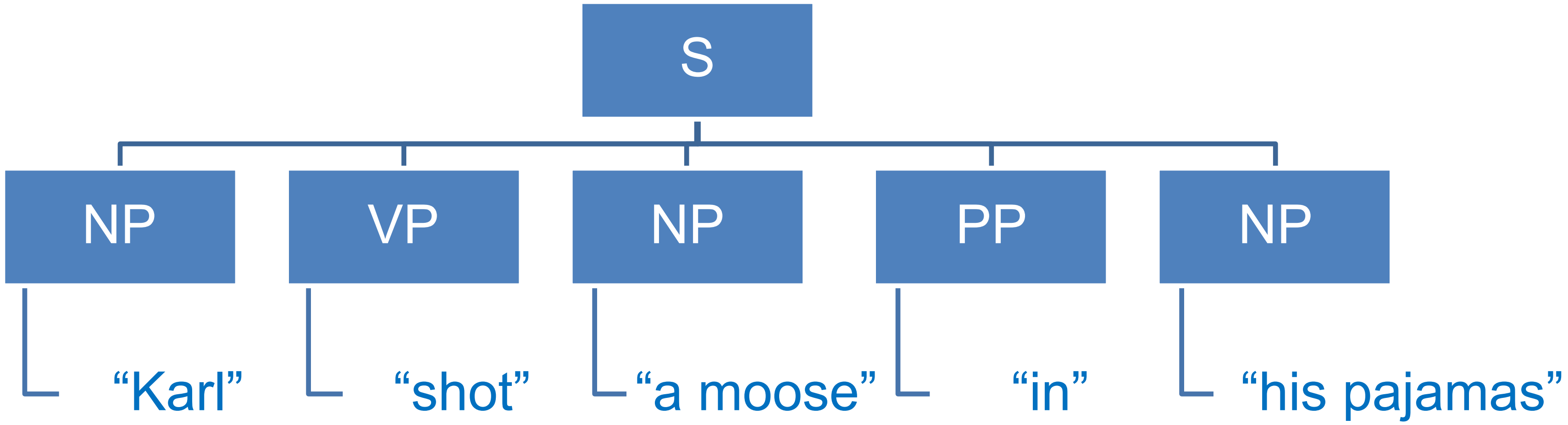
Ambiguity and Parse Trees

Who was wearing pajamas?



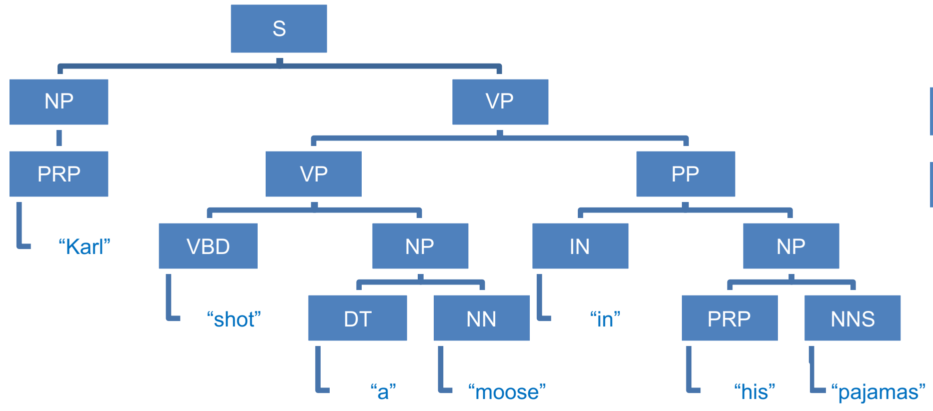
Ambiguity and Parse Trees

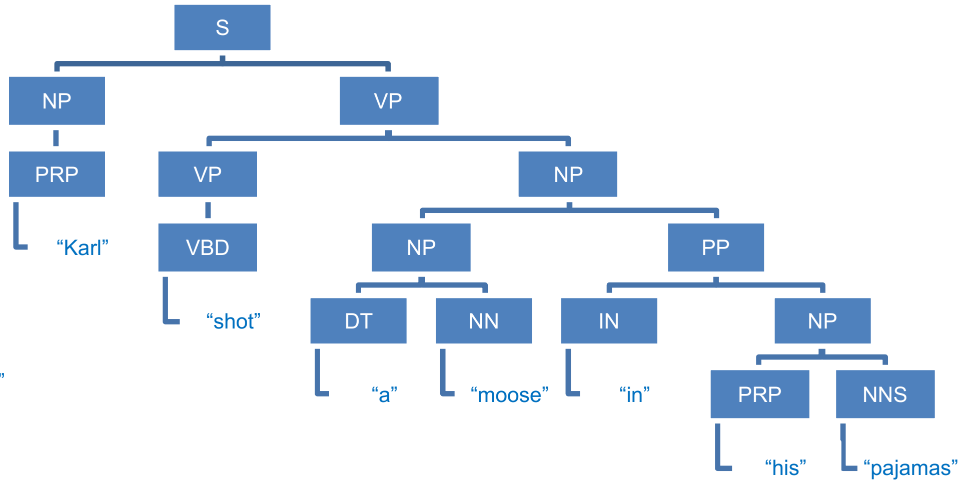
Notice the chunked version of both trees would be *the same.*



Ambiguity and Parse Trees

- This is a simple case—you can have longer sentences with even more alternative parse trees.
- A higher-level analysis involving semantics, pragmatics, or background knowledge would be needed to determine the "*right*" *(normative)* parse.





**probabilistic context-free grammar*

DataScience@SMU

Syntactic Analysis:

Full Grammar Parsing—Creating Parse Trees

Natural Language Processing

Creating Parse Trees

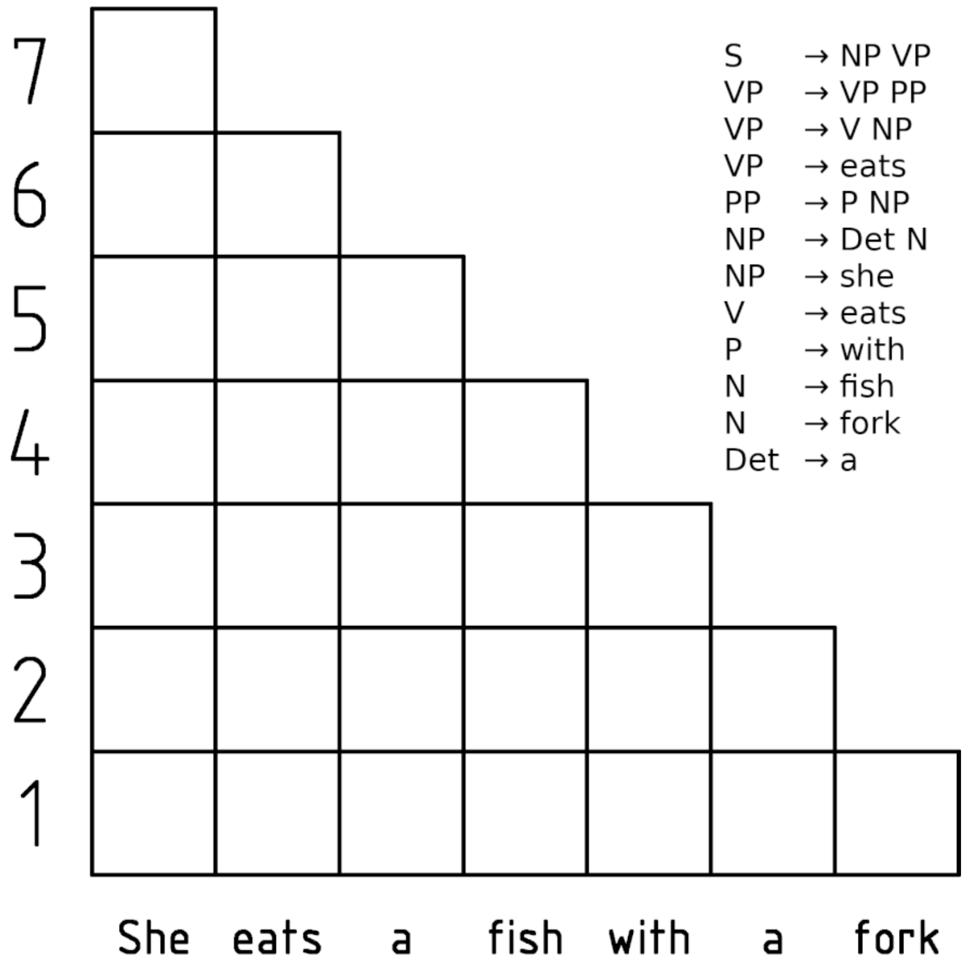
There are many different algorithms for creating parse trees, but perhaps the easiest to understand is the CYK* algorithm.

- It uses an increasingly large window to scan a sentence, labelling whether the whole window matches any existing grammar rule.
- Eventually the window widens to encompass the whole sentence and, hopefully, still matches the grammar!
- If so, we have a successful and complete parse of the sentence.

**Named after its inventors, John Cocke, Daniel Younger, and Tadao Kasami.*

CYK in Action

In this animation, notice how the procedure builds up levels of nested labels.



S	→ NP VP
VP	→ VP PP
VP	→ V NP
VP	→ eats
PP	→ P NP
NP	→ Det N
NP	→ she
V	→ eats
P	→ with
N	→ fish
N	→ fork
Det	→ a

CYK Detail

Focusing on a step…

$S \longrightarrow NP\ VP$

$VP \longrightarrow VP\ PP$

$\boxed{VP \longrightarrow V\ NP}$

$VP \longrightarrow \text{eats}$

$PP \longrightarrow P\ NP$

$NP \longrightarrow Det\ N$

$NP \longrightarrow \text{she}$

$V \longrightarrow \text{eats}$

$P \longrightarrow \text{with}$

$N \longrightarrow \text{fish}$

$N \longrightarrow \text{fork}$

$Det \longrightarrow \text{a}$

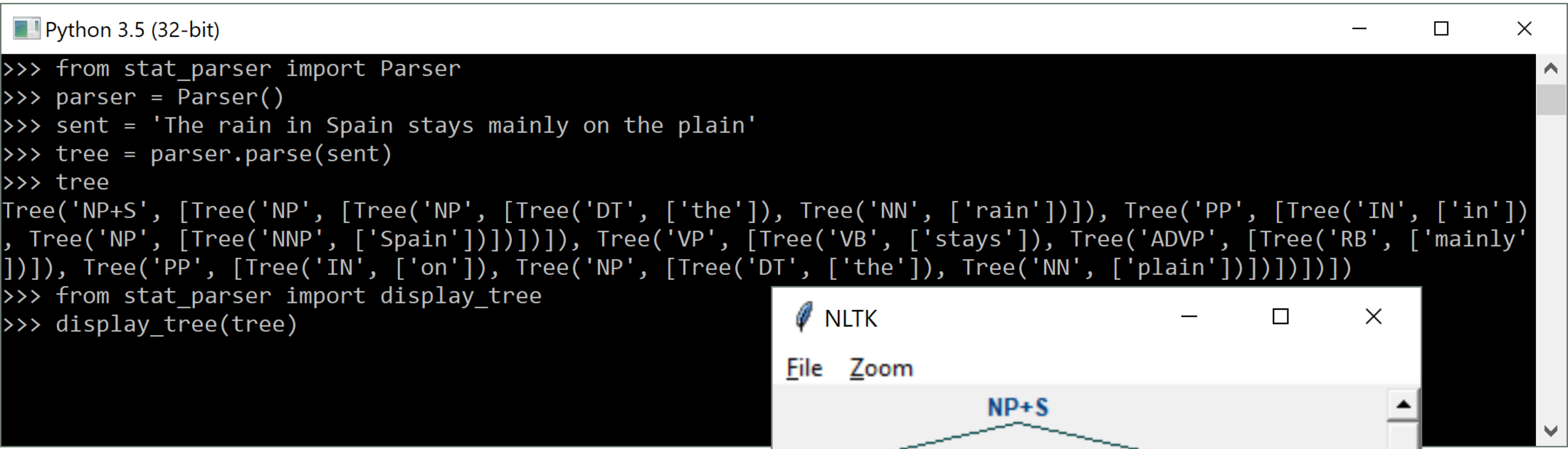
CYK table

7	**S**						
6		VP					
5							
4	**S**						
3		VP			PP		
2	**S**		NP			NP	
1	NP	V, VP	Det.	N	P	Det	N
	she	eats	a	fish	with	a	fork

Here a single rule in the grammar fills the three-word window exactly, by catching items *below* in the table.

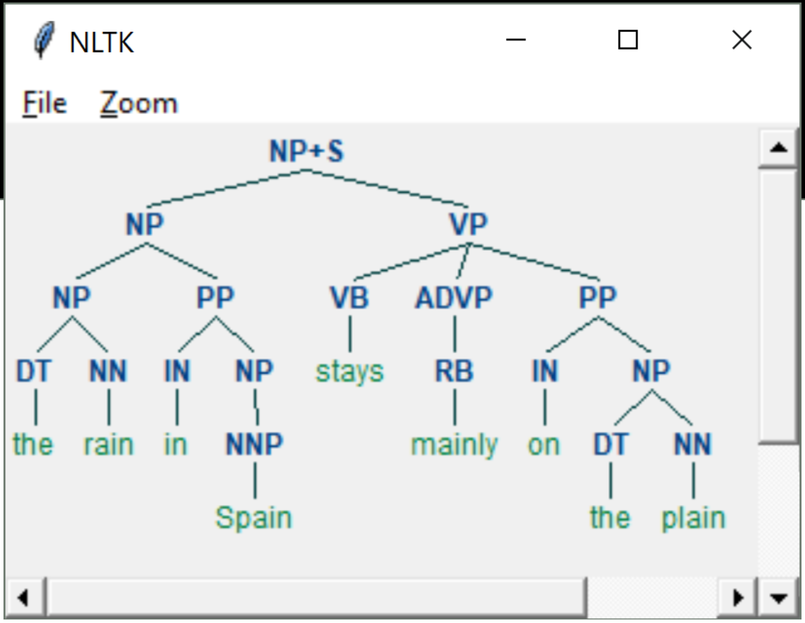
CYK for Everybody

It’s this simple to invoke CYK in Python:



Python 3.5 (32-bit)

```
>>> from stat_parser import Parser
>>> parser = Parser()
>>> sent = 'The rain in Spain stays mainly on the plain'
>>> tree = parser.parse(sent)
>>> tree
Tree('NP+S', [Tree('NP', [Tree('NP', [Tree('DT', ['the']), Tree('NN', ['rain'])]), Tree('PP', [Tree('IN', ['in'])
, Tree('NP', [Tree('NNP', ['Spain'])])])]), Tree('VP', [Tree('VB', ['stays']), Tree('ADVP', [Tree('RB', ['mainly'
])]), Tree('PP', [Tree('IN', ['on']), Tree('NP', [Tree('DT', ['the']), Tree('NN', ['plain'])])])])])
>>> from stat_parser import display_tree
>>> display_tree(tree)
```



- *This uses pyStatParser (available on GitHub), which extends CYK by adding weights to the grammar rules, i.e., it uses a PCFG.*
- *It can then take the highest-probability tree if it finds that more than one tree is possible.*

Other Parsers to Consider

There are plenty of parsers out there, and here are a few suggestions:

- Dependency parsing
  - MST parser
  - MALT parser
- Constituency parsing
  - Stanford parser
  - Link grammar parser

DataScience@SMU

Syntactic Analysis: Uses for Parse Trees

Natural Language Processing

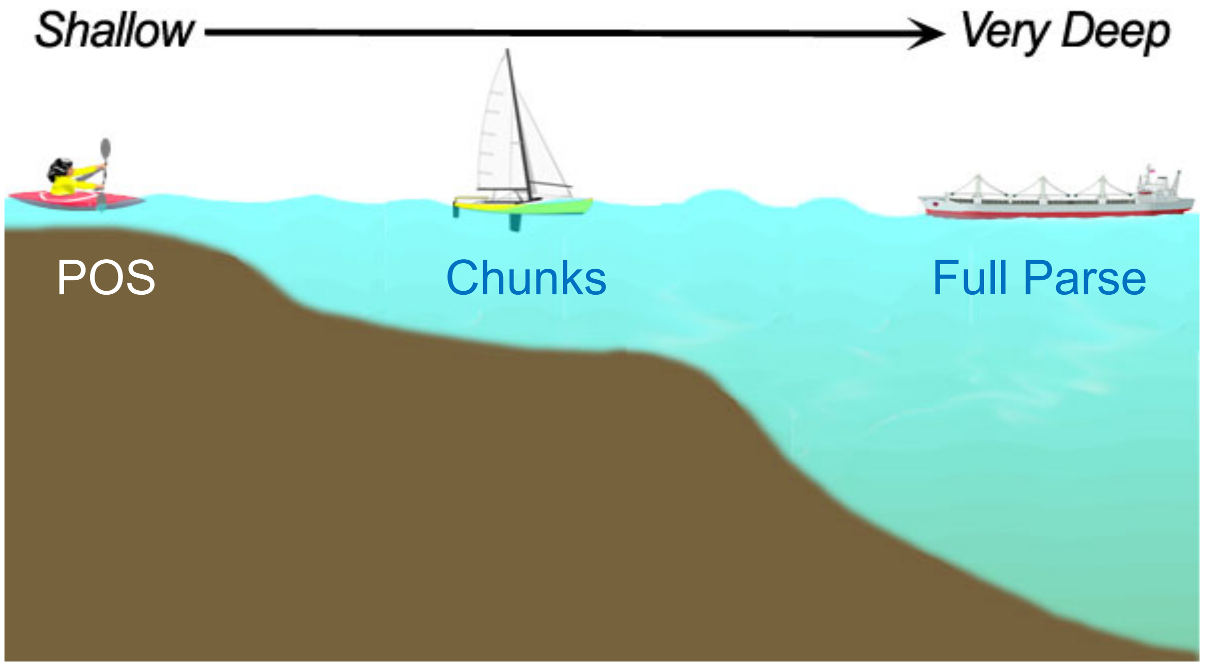
DataScience@SMU

Deep vs. Shallow Parsing

Shallow (or light) parsing gives us useful chunks, avoids some of the ambiguities of deep parsing, and runs faster.

- Strictly speaking, POS tagging is the "shallowest" parsing.

Deep parsing is required where we need to have finesse with the *relations between subphrases* in a sentence.



Dependency vs. Constituency

Which one is best for you? It depends on:

- The application
  - Text mining applications often use constituency parsers as a setup for information extraction.
  - Question-answering applications often use a dependency parser to validate answers.

Dependency vs. Constituency

Which one is best for you? It depends on:

- The language
  - Both kinds of parsers are great in English, German, and other languages having fairly strict word-order rules.
  - Dependency parsers are easier to engineer for MoR-FWO* languages such as Czech, Turkish, and Hindi, and for “resource-poor languages” (where there’s a lack of annotated corpora).

**morphologically rich free word order*

Dependency Parsers in Question Answering

- Why would dependency parsers come in handy for validating answers to natural language questions?
- Let’s take this example:

```
“Who robbed the soldiers?
```

Dependency Parsers in Question Answering

```
"Who robbed the soldiers?
```

Search hits on **content words** from the question:

"… two tourists were allegedly **robbed** by **soldiers** at gunpoint…."

"… **soldiers robbed** each other's dance partners as the night's festivities continued…"

"… the **soldiers** were **robbed** of their dignity when the captain made them clean the kennels and outhouses…"

"… the thief **robbed** three **soldiers** of their boots while they slept…"

Dependency Parsers in Question Answering

"… two tourists were allegedly **robbed** by **soldiers** at gunpoint…."

"… **soldiers robbed** each other's dance partners as the night's festivities continued …"

"… the **soldiers** were **robbed** of their dignity when the captain made them clean the kennels and outhouses…"

"… the thief **robbed** three **soldiers** of their boots while they slept…"

Running a dependency parser is a good way to tell which hits have "soldiers" as *the object* of "robbed," not the subject.

Constituency Parsers

But constituency parsers are more convenient to us for grabbing structured phrases at various levels of atomicity …handy for highlighting and summarization. Consider:

```
"In celebration of the Olympic victory, their age-
old disputes over centralization were put on hold."
```

If we want various-sized "snippets" from this sentence that *make sense as phrases*, a constituency parser is a good tool to glean the following:

```
"victory"
"Olympic victory"
"celebration of the Olympic victory"
"disputes"
"disputes over centralization"
"age-old disputes over centralization"
```

Super valuable for autogenerating metadata keywords for web pages, word clouds for documents, document summaries, etc.

DataScience@SMU

Syntactic Analysis:
Combining Lexical and Syntactic Analysis

Natural Language Processing

DataScience@SMU

Combining Lexical and Syntactic Analysis

If we combine a lexical KB* with syntax parsing, we can make improvements in:

- Information extraction
- Sentiment analysis

**lexical knowledge base*

Improving Information Extraction

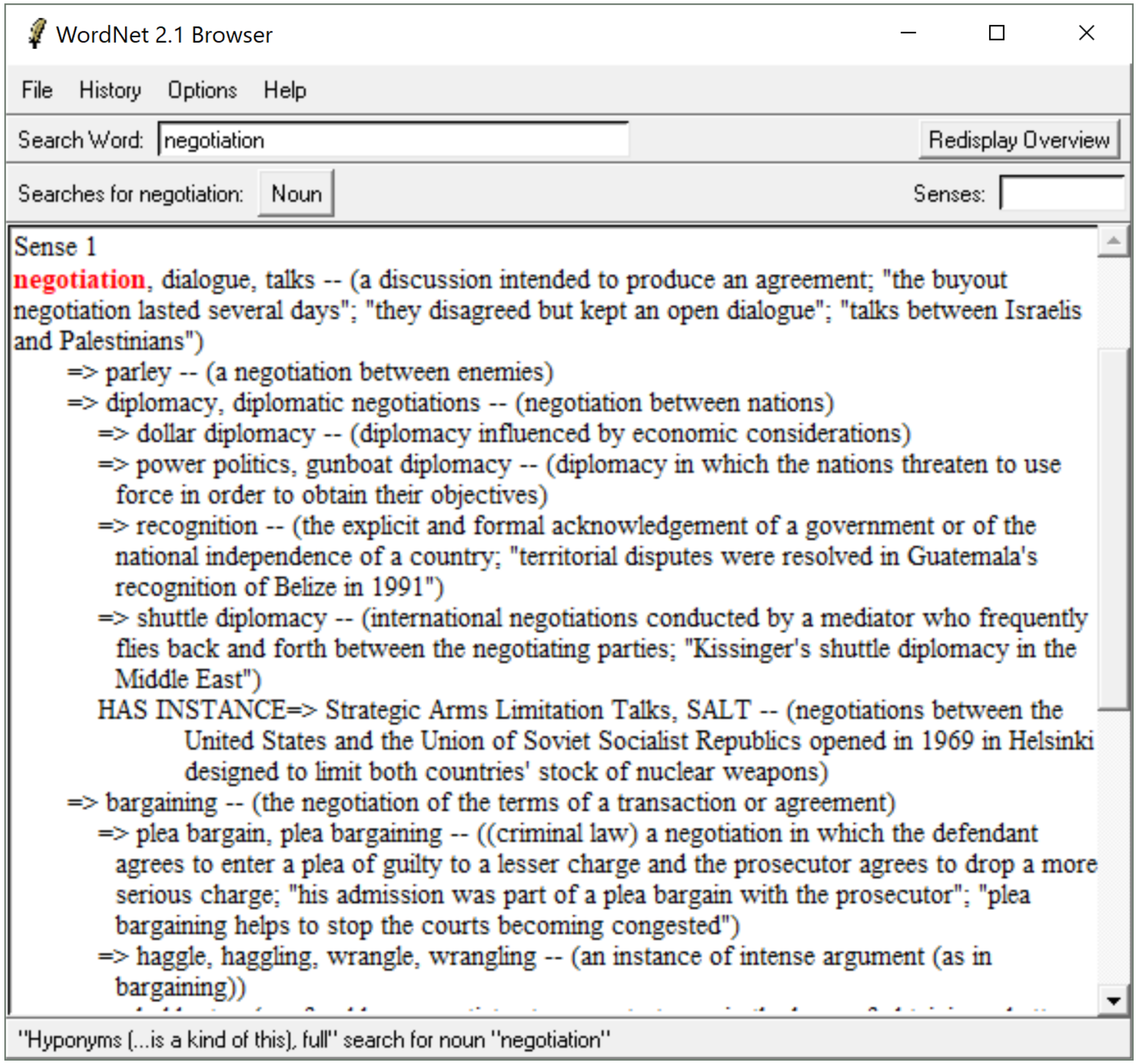
Topic Definition and Processing

1. Given a headword at a middle level of ontology, create a word class using hyponym or troponym trees—this defines a topic.
2. Find phrases that exemplify the topic by either:
    1. Using a constituency parser to find NPs, VPs led by a member of the word class, or
    2. Using a dependency parser to find objects of verbs or modifier of nouns
3. Now use frequency to gather the most prominent phrases embodying our topic.

Information Extraction Example

Topic: Negotiation

N1: negotiation, has 2 synonyms and 20 hyponyms



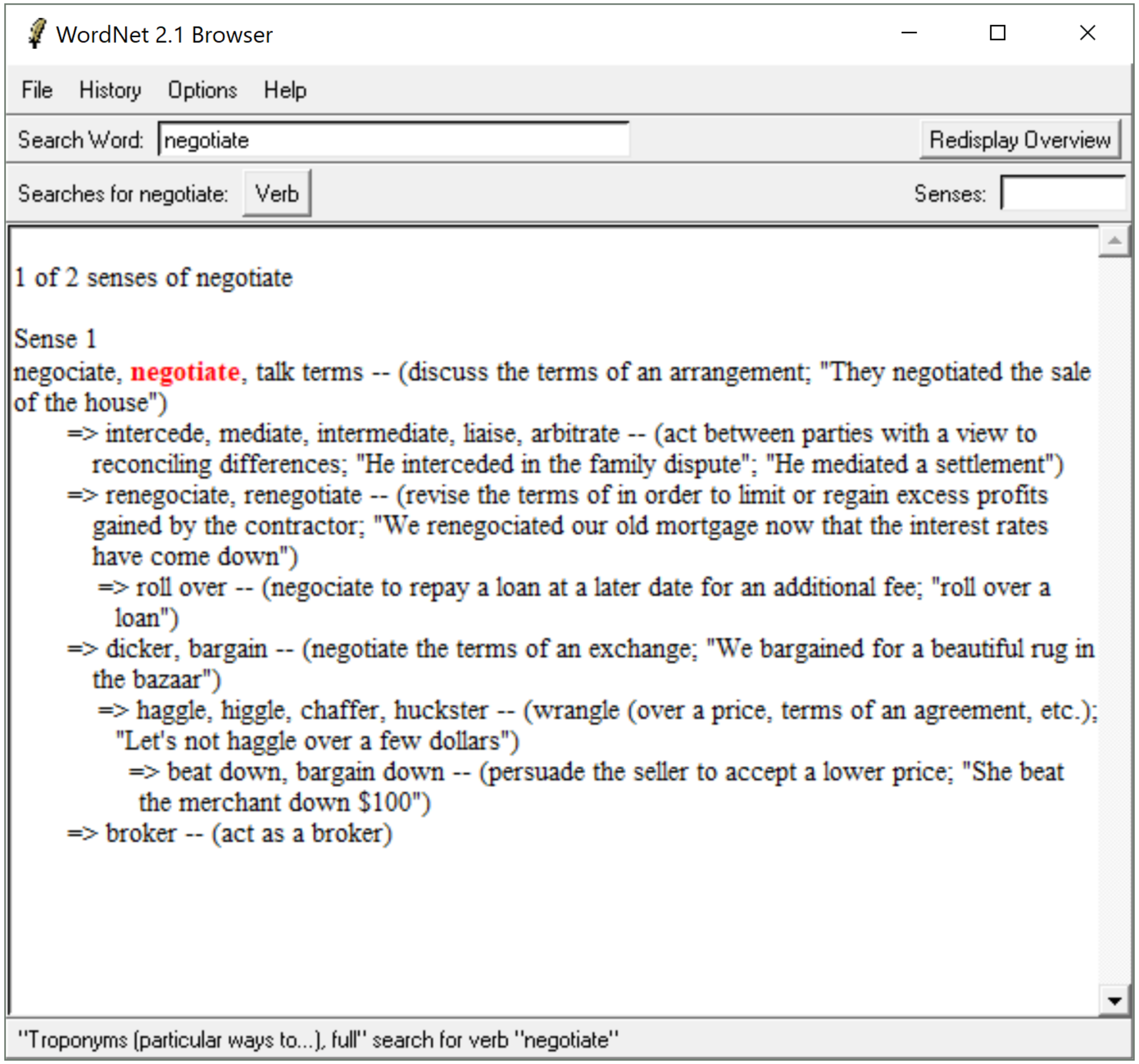
```
Sense 1
negotiation, dialogue, talks -- (a discussion intended to produce an agreement; "the buyout
negotiation lasted several days"; "they disagreed but kept an open dialogue"; "talks between Israelis
and Palestinians")
       => parley -- (a negotiation between enemies)
       => diplomacy, diplomatic negotiations -- (negotiation between nations)
           => dollar diplomacy -- (diplomacy influenced by economic considerations)
           => power politics, gunboat diplomacy -- (diplomacy in which the nations threaten to use
              force in order to obtain their objectives)
           => recognition -- (the explicit and formal acknowledgement of a government or of the
              national independence of a country; "territorial disputes were resolved in Guatemala's
              recognition of Belize in 1991")
           => shuttle diplomacy -- (international negotiations conducted by a mediator who frequently
              flies back and forth between the negotiating parties; "Kissinger's shuttle diplomacy in the
              Middle East")
           HAS INSTANCE=> Strategic Arms Limitation Talks, SALT -- (negotiations between the
                     United States and the Union of Soviet Socialist Republics opened in 1969 in Helsinki
                     designed to limit both countries' stock of nuclear weapons)
       => bargaining -- (the negotiation of the terms of a transaction or agreement)
           => plea bargain, plea bargaining -- ((criminal law) a negotiation in which the defendant
              agrees to enter a plea of guilty to a lesser charge and the prosecutor agrees to drop a more
              serious charge; "his admission was part of a plea bargain with the prosecutor"; "plea
              bargaining helps to stop the courts becoming congested")
           => haggle, haggling, wrangle, wrangling -- (an instance of intense argument (as in
              bargaining))
```

"Hyponyms (...is a kind of this), full" search for noun "negotiation"

Information Extraction Example

Topic: Negotiation

V1: negotiate, has 2 synonyms and 17 hyponyms



1 of 2 senses of negotiate

Sense 1
negociate, **negotiate**, talk terms -- (discuss the terms of an arrangement; "They negotiated the sale of the house")
    => intercede, mediate, intermediate, liaise, arbitrate -- (act between parties with a view to reconciling differences; "He interceded in the family dispute"; "He mediated a settlement")
    => renegociate, renegotiate -- (revise the terms of in order to limit or regain excess profits gained by the contractor; "We renegociated our old mortgage now that the interest rates have come down")
        => roll over -- (negociate to repay a loan at a later date for an additional fee; "roll over a loan")
    => dicker, bargain -- (negotiate the terms of an exchange; "We bargained for a beautiful rug in the bazaar")
        => haggle, higgle, chaffer, huckster -- (wrangle (over a price, terms of an agreement, etc.); "Let's not haggle over a few dollars")
            => beat down, bargain down -- (persuade the seller to accept a lower price; "She beat the merchant down $100")
    => broker -- (act as a broker)

"Troponyms (particular ways to...), full" search for verb "negotiate"

Information Extraction Example

Following our procedure of capturing dependent modifiers of a noun, or dependent objects of a verb (or constituents of NPs and VPs led by of our headwords), we yield:

```
Topic: Negotiation

“drawn-out negotiations”

“beat down the price”

“renegotiate the mortgage”

“intense haggling”
```

Improving Sentiment Analysis

- Basic sentiment analysis can be done simply with a special lexical KB having *valences* for a subset of our domain vocabulary.
- A *valence* is a number, usually normalized from -1 to 1, indicating if the word is emotive, positively or negatively.

Word	Valence
awkward	-0.5
delightful	0.6
disgusted	-0.9
dislike	-0.5
exuberant	0.7
hate	-0.7
like	0.5
love	0.8
miserable	-0.8
revolting	-0.9
rough	-0.3
smooth	0.3
uplifting	0.5
yucky	-0.5

Improving Sentiment Analysis

- Our default sentiment analysis is simply to score each sentence with its cumulative valence score.
- There is an obvious weakness: we can't tell what the sentiment is *about.*

Word	Valence
awkward	-0.5
delightful	0.6
disgusted	-0.9
dislike	-0.5
exuberant	0.7
hate	-0.7
like	0.5
love	0.8
miserable	-0.8
revolting	-0.9
rough	-0.3
smooth	0.3
uplifting	0.5
yucky	-0.5

Improving Sentiment Analysis

Taken as a whole, this sentence is going to register zero net valence (-0.5 + 0.5 = 0).

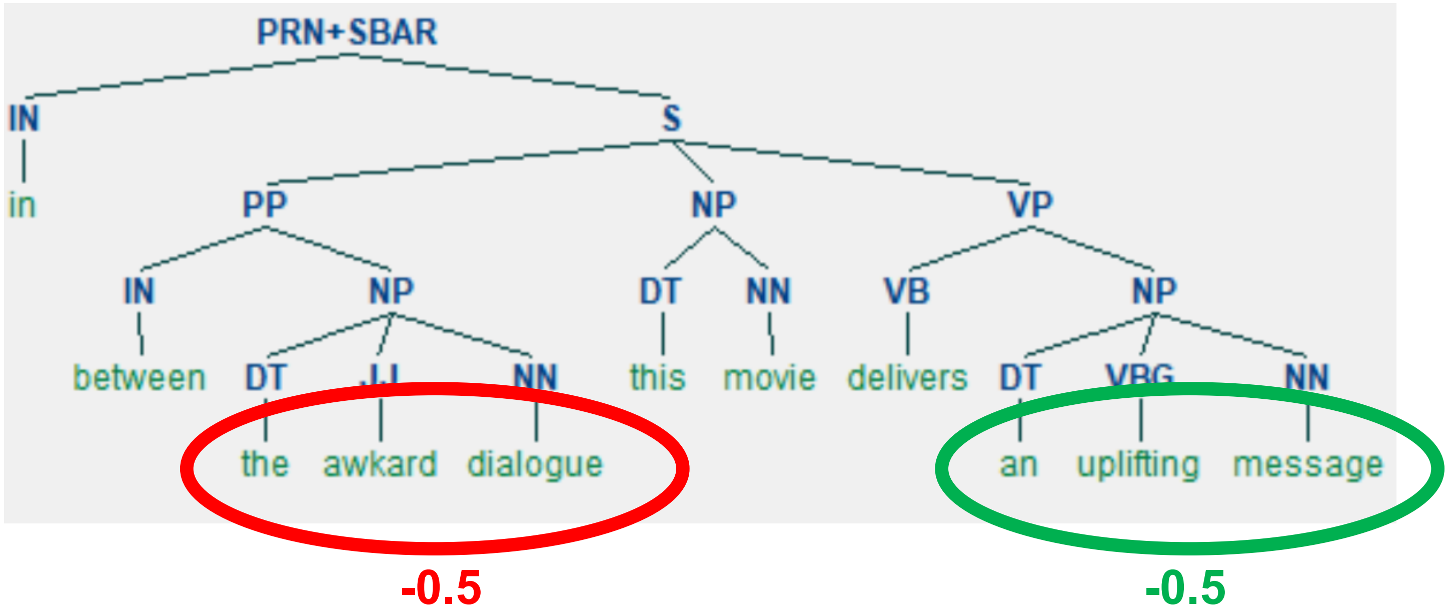
```
"In between the awkward dialogue this movie delivers an uplifting message."
```

But we could use either a dependency or a constituency parser, or even a chunker, to make this a little better.

Improving Sentiment Analysis

"In between the **awkward** dialogue this movie delivers an **uplifting** message."

With a constituency parse, we can separate the sentiment triggers, outputting intelligible phrases (having explanatory power).



Looking at this, you can tell that a chunker could have done the same work for us.

DataScience@SMU