What is spaCy?

SpaCy is a **free, open-source library** for advanced **Natural Language Processing** (NLP) in Python.

If we're working with a lot of text, we'll eventually want to know more about it. For example, what's it about? What do the words mean in context? Who is doing what to whom? What companies and products are mentioned? Which texts are similar to each other?

spaCy is designed specifically for **production use** and helps you build applications that process and "understand" large volumes of text. It can be used to build **information extraction or natural language understanding** systems, or to pre-process text for **deep learning**.

In [1]:

```
#Install spaCy
#spaCy is compatible with 64-bit CPython 2.7 / 3.5+
#and runs on Unix/Linux, macOS/OS X and Windows.

!pip install -U spacy
```

...

Models & Languages

- Install a model
- get the code to load it from within spaCy

In [40]:

```
#german
!python -m spacy download de_core_news_sm

#english
!python -m spacy download en_core_web_sm

!python -m spacy download en_core_web_md
```

...

Features

spaCy's features and capabilities.

NAME	DESCRIPTION
Tokenization	Segmenting text into words, punctuations marks etc.
Part-of-speech (POS) **Tagging**	Assigning word types to tokens, like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Lemmatization	Assigning the base forms of words. For example, the lemma of "was" is "be", and the lemma of "rats" is "rat".
Sentence Boundary Detection (SBD)	Finding and segmenting individual sentences.
Named Entity Recognition (NER)	Labelling named "real-world" objects, like persons, companies or locations.
Entity Linking (EL)	Disambiguating textual entities to unique identifiers in a Knowledge Base.
Similarity	Comparing words, text spans and documents and how similar they are to each other.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Rule-based Matching	Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions.
Training	Updating and improving a statistical model's predictions.
Serialization	Saving objects to files or byte strings.

1. Linguistic annotations

spaCy provides a variety of linguistic annotations to give us **insights into a text's grammatical structure**. This includes the word types, like the parts of speech, and how the words are related to each other. For example, if you're analyzing text, it makes a huge difference whether a noun is the subject of a sentence, or the object – or whether "google" is used as a verb, or refers to the website or company in a specific context.

In [42]:

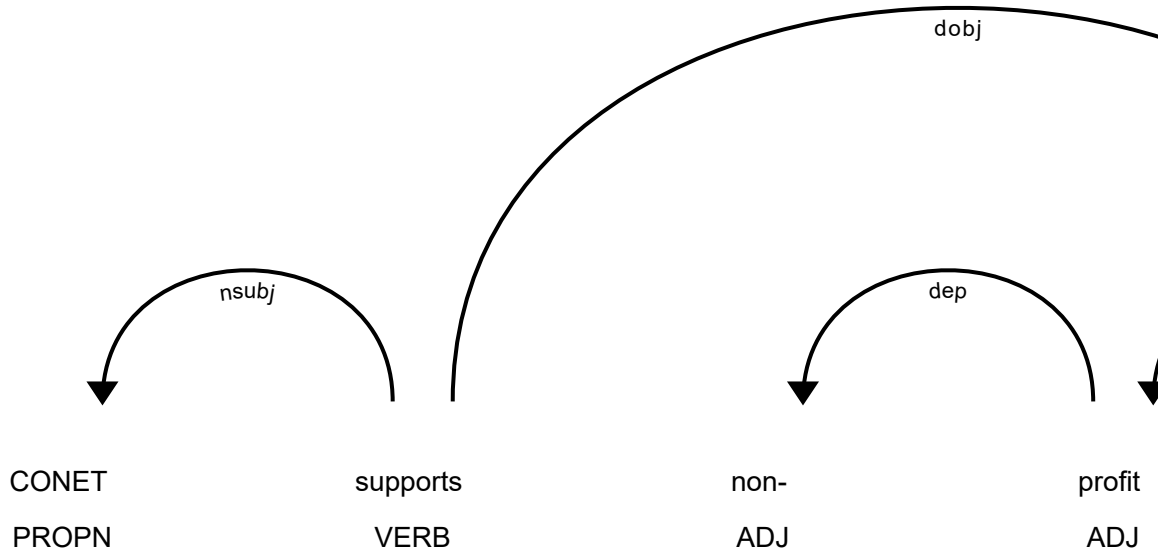
```
import spacy

#loading models (english)
nlp = spacy.load("en_core_web_sm")


doc = nlp("CONET supports non-profit organizations in the region.")

#Text: The original word text.
#Lemma: The base form of the word.
#POS: The simple UPOS part-of-speech tag.
#Tag: The detailed part-of-speech tag.
#Dep: Syntactic dependency, i.e. the relation between tokens.
#Shape: The word shape – capitalization, punctuation, digits.
#is alpha: Is the token an alpha character?
#is stop: Is the token part of a stop list, i.e. the most common words of the language?

for token in doc:
    print(token.text, token.pos_, token.dep_)
```

```
CONET PROPN nsubj
supports VERB ROOT
non ADJ dep
- ADJ dep
profit ADJ amod
organizations NOUN dobj
in ADP prep
the DET det
region NOUN pobj
. PUNCT punct
```

Visualization

In [43]:

```
#displacy is a built-in visualization suite.
from spacy import displacy

#options = {"compact": True, "color": "blue"}
#displacy.serve(doc, style="dep", options=options)

displacy.render(doc, style="dep")
```



In [8]:

```
import spacy

#loading models (german)
nlp = spacy.load("de_core_news_sm")

doc = nlp("CONET unterstützt gemeinnützige Organisationen in der Region.")

#Text: The original word text.
#Lemma: The base form of the word.
#POS: The simple UPOS part-of-speech tag.
#Tag: The detailed part-of-speech tag.
#Dep: Syntactic dependency, i.e. the relation between tokens.
#Shape: The word shape - capitalization, punctuation, digits.
#is alpha: Is the token an alpha character?
#is stop: Is the token part of a stop list, i.e. the most common words of the language?

for token in doc:
    print(token.text, token.pos_, token.dep_)
```

```
CONET PROPN sb
unterstützt VERB ROOT
gemeinnützige ADJ nk
Organisationen NOUN oa
in ADP mnr
der DET nk
Region NOUN nk
. PUNCT punct
```

* Tokenization

During processing, spaCy first **tokenizes** the text, i.e. segments it into words, punctuation and so on. This is done by applying rules specific to each language. For example, punctuation at the end of a sentence should be split off – whereas “U.K.” should remain one token. Each Doc consists of individual tokens, and we can iterate over them:

In [10]:

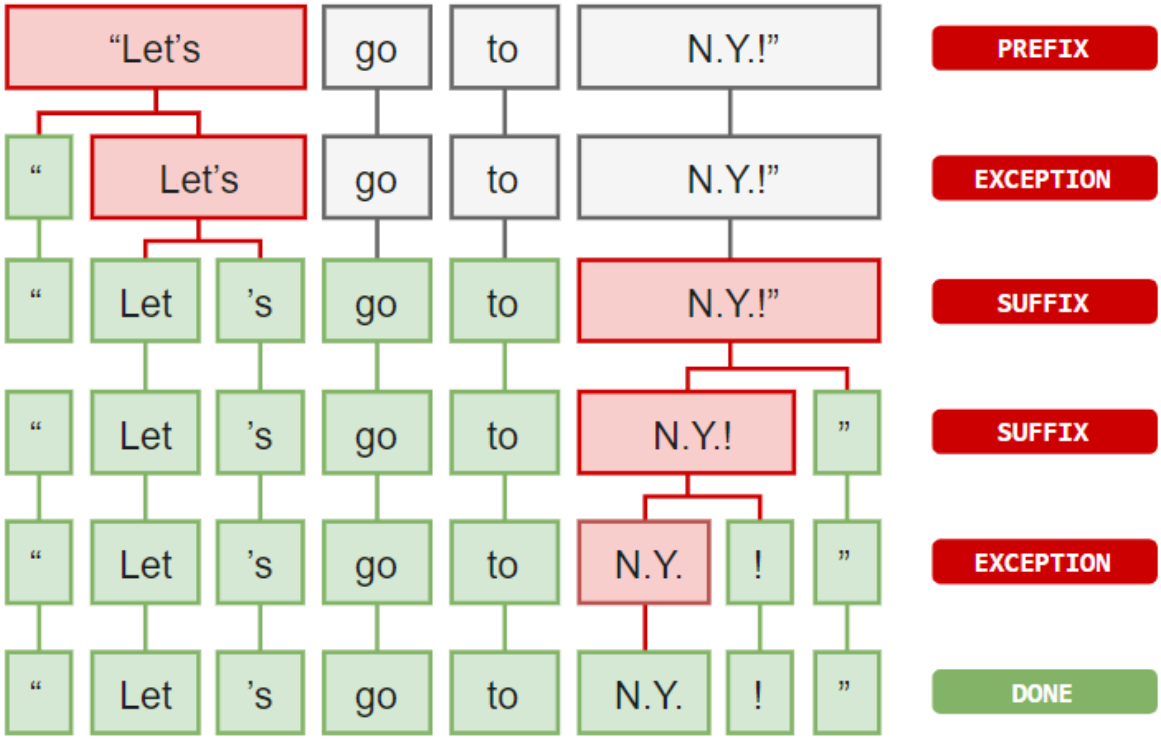
```
doc = nlp("CONET supports non-profit organizations in the region.")
for token in doc:
    print(token.text)
```

```
CONET
supports
non-profit
organizations
in
the
region
.
```

First, the raw text is split on whitespace characters, similar to text.split(' '). Then, the tokenizer processes the text from left to right. On each substring, it performs two checks:

1. **Does the substring match a tokenizer exception rule?** For example, “don’t” does not contain whitespace, but should be split into two tokens, “do” and “n’t”, while “U.K.” should always remain one token.

2. **Can a prefix, suffix or infix be split off?** For example punctuation like commas, periods, hyphens or quotes. If there's a match, the rule is applied and the tokenizer continues its loop, starting with the newly split substrings. This way, spaCy can split complex, nested tokens like combinations of abbreviations and multiple punctuation marks.



- Tokenizer exception: Special-case rule to split a string into several tokens or prevent a token from being split when punctuation rules are applied.
- Prefix: Character(s) at the beginning, e.g. $, (, “, ¿.
- Suffix: Character(s) at the end, e.g. km,), ”, !.
- Infix: Character(s) in between, e.g. -, --, /, ….

* Part-of-speech tags and dependencies

After tokenization, spaCy can **parse** and **tag** a given Doc. This is where the statistical model comes in, which enables spaCy to make a prediction of which tag or label most likely applies in this context. A model consists of binary data and is produced by showing a system enough examples for it to **make predictions** that generalize across the language – for example, a word following “the” in English is most likely a noun.

- Text: The original word text.
- Lemma: The base form of the word.
- POS: The simple UPOS part-of-speech tag.
- Tag: The detailed part-of-speech tag.
- Dep: Syntactic dependency, i.e. the relation between tokens.
- Shape: The word shape – capitalization, punctuation, digits.
- is alpha: Is the token an alpha character?
- is stop: Is the token part of a stop list, i.e. the most common words of the language?

In [44]:

```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("The Hennef IT Consulting Company CONET donates 8,100 euros to socially committed

for token in doc:
    print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
            token.shape_, token.is_alpha, token.is_stop)
```

```
The the DET DT det Xxx True True
Hennef Hennef PROPN NNP compound Xxxxx True False
IT IT PROPN NNP compound XX True True
Consulting Consulting PROPN NNP compound Xxxxx True False
Company Company PROPN NNP compound Xxxxx True False
CONET CONET PROPN NNP nsubj XXXX True False
donates donate VERB VBZ ROOT xxxx True False
8,100 8,100 NUM CD nummod d,ddd False False
euros euro NOUN NNS dobj xxxx True False
to to PART TO aux xx True True
socially socially ADV RB advmod xxxx True False
committed commit VERB VBN xcomp xxxx True False
institutions institution NOUN NNS dobj xxxx True False
and and CCONJ CC cc xxx True True
associations association NOUN NNS conj xxxx True False
. . PUNCT . punct . False False
```

In [45]:

```
#visualization
options = {"compact": True, "bg": "#09a3d5",
           "color": "white", "font": "Source Sans Pro"}
displacy.render(doc, style="dep", options=options)
```

* Named Entities

A named entity is a “real-world object” that’s assigned a name – for example, a person, a country, a product or a book title. spaCy can **recognize** various types of named entities in a document, by asking the model for a **prediction**. Because models are statistical and strongly depend on the examples they were trained on.

In [31]:

```
import spacy

nlp = spacy.load("en_core_web_sm")

text = 'The Hennef IT Consulting Company CONET donates €8,100 euros to socially committed i

doc = nlp(text)

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
The Hennef IT Consulting Company 0 32 ORG
8,100 48 53 MONEY
```

In [39]:

```
# Visualizing the entity recognizer
# ORG: Companies, agencies, institutions.
# Money: Monetary values, including unit.
# GPE: Geopolitical entity, i.e. countries, cities, states.

import spacy
from spacy import displacy

text1 = "CONET donates €8,100 euros to socially committed institutions and associations."

nlp = spacy.load("en_core_web_sm")

doc1 = nlp(text1)

displacy.render(doc1, style="ent")
```

CONET ORG donates € 8,100 MONEY euros to socially committed institutions and associations.

In [38]:

```
# Visualizing the entity recognizer
# ORG: Companies, agencies, institutions.
# Money: Monetary values, including unit.
# GPE: Geopolitical entity, i.e. countries, cities, states.

text2 = "Apple is looking at buying U.K. startup for $1 billion"

doc2 = nlp(text2)

displacy.render(doc2, style="ent")
```

Apple **ORG** is looking at buying U.K. **GPE** startup for $1 billion **MONEY**

* Word vectors and similarity

Similarity is determined by comparing word vectors or “word embeddings”, multi-dimensional meaning representations of a word. Word vectors can be generated using an algorithm like word2vec and usually look like this:

In [2]:

```
import spacy

nlp = spacy.load("en_core_web_md")
tokens = nlp("dog cat banana afskfsd")

for token in tokens:
    print(token.text, token.has_vector, token.vector_norm, token.is_oov)

#Text: The original token text.
#has vector: Does the token have a vector representation?
#Vector norm: The L2 norm of the token’s vector (the square root of the sum of the values s
#OOV: Out-of-vocabulary

# we can use 'en_vectors_web_lg', which includes over 1 million unique vectors. (large voca
```

```
dog True 7.0336733 False
cat True 6.6808186 False
banana True 6.700014 False
afskfsd False 0.0 True
```

In [8]:

```
#able to compare two objects, and make a prediction of how similar they are.
#Predicting similarity is useful for building recommendation systems or flagging duplicates

import spacy

nlp = spacy.load("en_core_web_md")
tokens = nlp("dog cat banana")

for token1 in tokens:
    for token2 in tokens:
        print(token1.text, token2.text, token1.similarity(token2))
```

```
dog dog 1.0
dog cat 0.80168545
dog banana 0.24327648
cat dog 0.80168545
cat cat 1.0
cat banana 0.28154367
banana dog 0.24327648
banana cat 0.28154367
banana banana 1.0
```

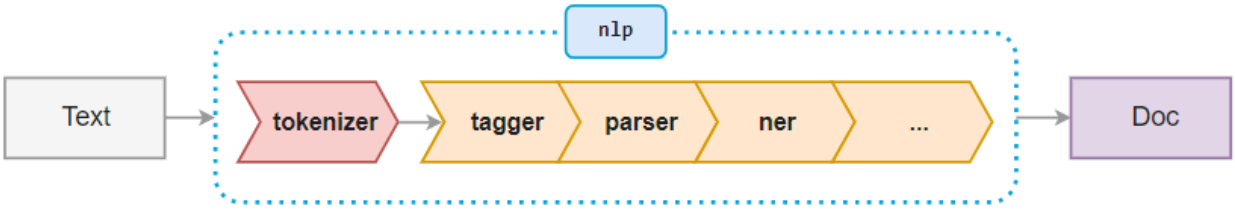
In [50]:

```
#more example

import spacy

nlp = spacy.load("en_core_web_md")
doc = nlp("Apple and banana are similar. Pasta and hippo aren't.")

apple = doc[0]
banana = doc[2]
pasta = doc[6]
hippo = doc[8]

print("apple <-> banana", apple.similarity(banana))
print("pasta <-> hippo", pasta.similarity(hippo))
print(apple.has_vector, banana.has_vector, pasta.has_vector, hippo.has_vector)
```

```
apple <-> banana 0.5831844
pasta <-> hippo 0.120697394
True True True True
```

2. Pipeline

When you call nlp on a text, spaCy first tokenizes the text to produce a Doc object. The Doc is then processed in several different steps – this is also referred to as the **processing pipeline**. The pipeline used by the default models consists of a tagger, a parser and an entity recognizer. Each pipeline component returns the processed Doc, which is then passed on to the next component.



NAME	COMPONENT	CREATES	DESCRIPTION
tokenizer	`Tokenizer`	`Doc`	Segment text into tokens.
tagger	`Tagger`	`Doc[i].tag`	Assign part-of-speech tags.
parser	`DependencyParser`	`Doc[i].head`, `Doc[i].dep`, `Doc.sents`, `Doc.noun_chunks`	Assign dependency labels.
ner	`EntityRecognizer`	`Doc.ents`, `Doc[i].ent_iob`, `Doc[i].ent_type`	Detect and label named entities.
textcat	`TextCategorizer`	`Doc.cats`	Assign document labels.
...	custom components	`Doc._.xxx`, `Token._.xxx`, `Span._.xxx`	Assign custom attributes, methods or properties.

In [49]:

```
import spacy

texts = [
    "Net income was $9.4 million compared to the prior year of $2.7 million.",
    "Revenue exceeded twelve billion dollars, with a loss of $1b.",
]

nlp = spacy.load("en_core_web_sm")
for doc in nlp.pipe(texts, disable=["tagger", "parser"]):
    # Do something with the doc here
    print([(ent.text, ent.label_) for ent in doc.ents])
```

```
[('$9.4 million', 'MONEY'), ('the prior year', 'DATE'), ('$2.7 million', 'MO
NEY')]
[('twelve billion dollars', 'MONEY'), ('1b', 'MONEY')]
```

3. Vocab, hashes and lexemes

Whenever possible, spaCy tries to store data in a vocabulary, the Vocab, that will be shared by multiple documents. To save memory, spaCy also encodes all strings to hash values – in this case for example, “coffee” has the hash 3197928453018144401. Entity labels like “ORG” and part-of-speech tags like “VERB” are also

encoded. Internally, spaCy only “speaks” in hash values.

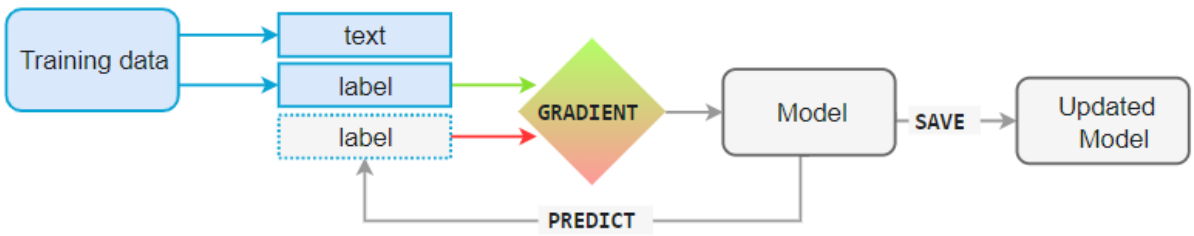
- Token: A word, punctuation mark etc. in context, including its attributes, tags and dependencies.
- Lexeme: A “word type” with no context. Includes the word shape and flags, e.g. if it’s lowercase, a digit or punctuation.
- Doc: A processed container of tokens in context.
- Vocab: The collection of lexemes.
- StringStore: The dictionary mapping hash values to strings, for example 3197928453018144401 → “coffee”.

In [48]:

```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("I love coffee")

coffee_hash = nlp.vocab.strings["coffee"]  # 3197928453018144401
coffee_text = nlp.vocab.strings[coffee_hash]  # 'coffee'
print(coffee_hash, coffee_text)
print(doc[2].orth, coffee_hash)  # 3197928453018144401
print(doc[2].text, coffee_text)  # 'coffee'

beer_hash = doc.vocab.strings.add("beer")  # 3073001599257881079
beer_text = doc.vocab.strings[beer_hash]  # 'beer'
print(beer_hash, beer_text)

unicorn_hash = doc.vocab.strings.add("🦄")  # 18234233413267120783
unicorn_text = doc.vocab.strings[unicorn_hash]  # '🦄'
print(unicorn_hash, unicorn_text)
```

```
3197928453018144401 coffee
3197928453018144401 3197928453018144401
coffee coffee
3073001599257881079 beer
18234233413267120783 🦄
```

4. Training



- Training data: Examples and their annotations.
- Text: The input text the model should predict a label for.
- Label: The label the model should predict.
- Gradient: Gradient of the loss function calculating the difference between input and expected output.

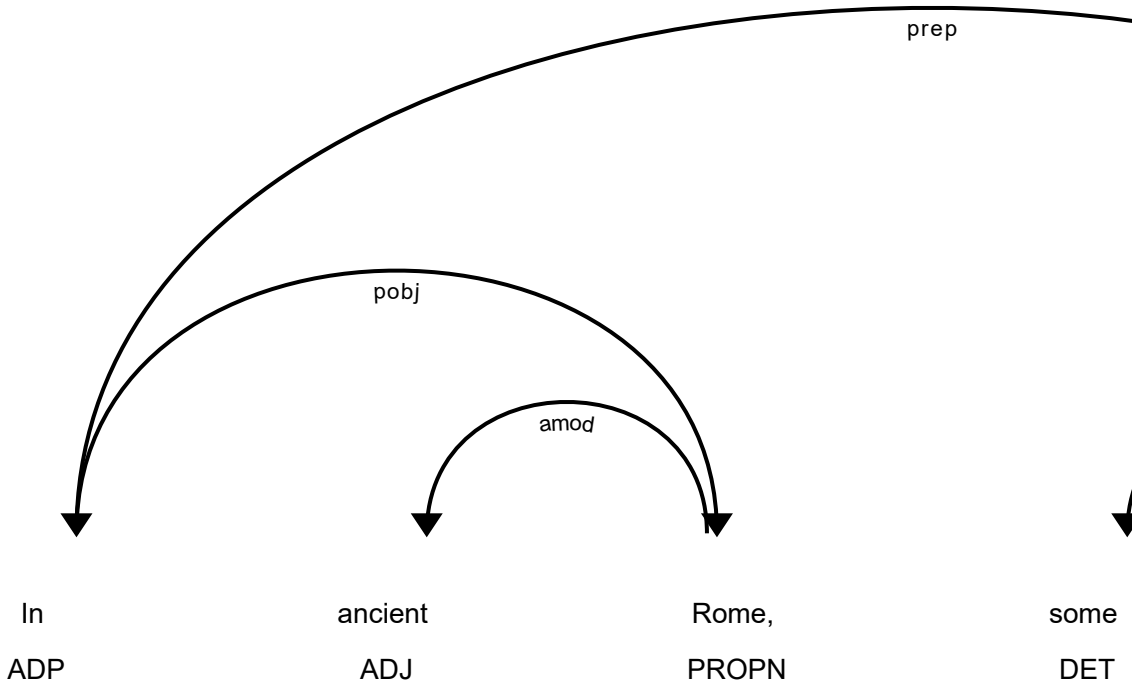
In [10]:

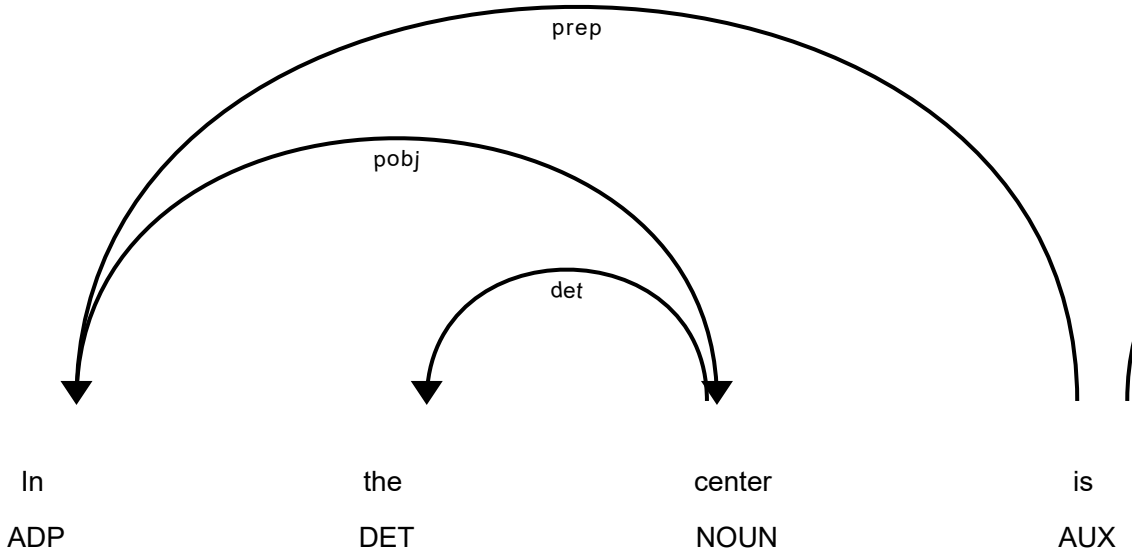
```
import spacy
import random

nlp = spacy.load("en_core_web_sm")
train_data = [("Uber blew through $1 million", {"entities": [(0, 4, "ORG")]})]

other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]
#load the model
with nlp.disable_pipes(*other_pipes):

    #begin_training(): Start the training and return an optimizer function to update the mo
    #Can take an optional function converting the training data to spaCy's training format.
    optimizer = nlp.begin_training()
    for i in range(10):
        random.shuffle(train_data)
        for text, annotations in train_data:
            #update(): Update the model with the training example and annotations data.
            nlp.update([text], [annotations], sgd=optimizer)
nlp.to_disk("/model") #Save the updated model to a directory.
```

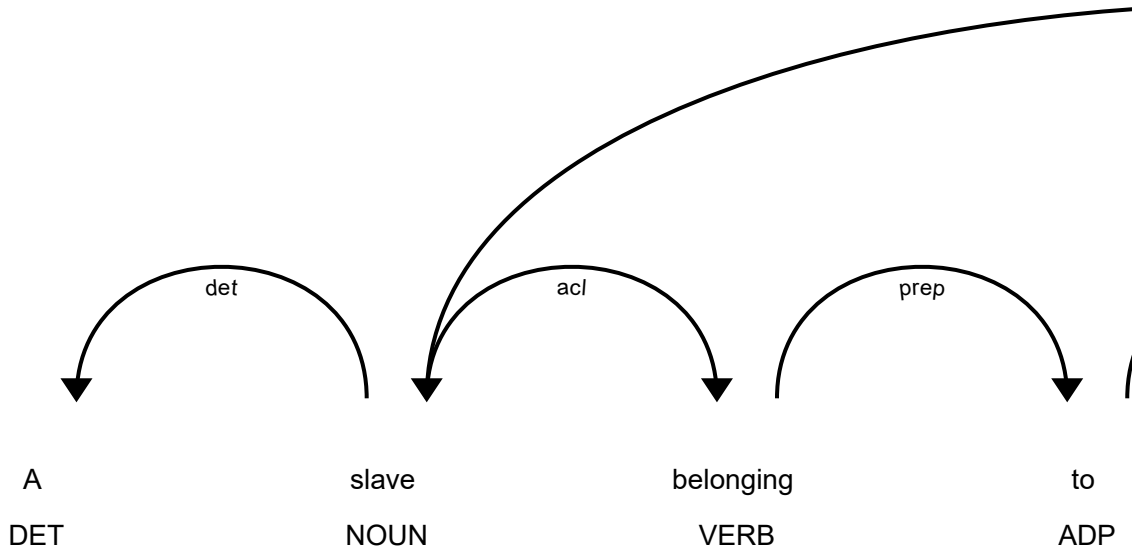
Visualization

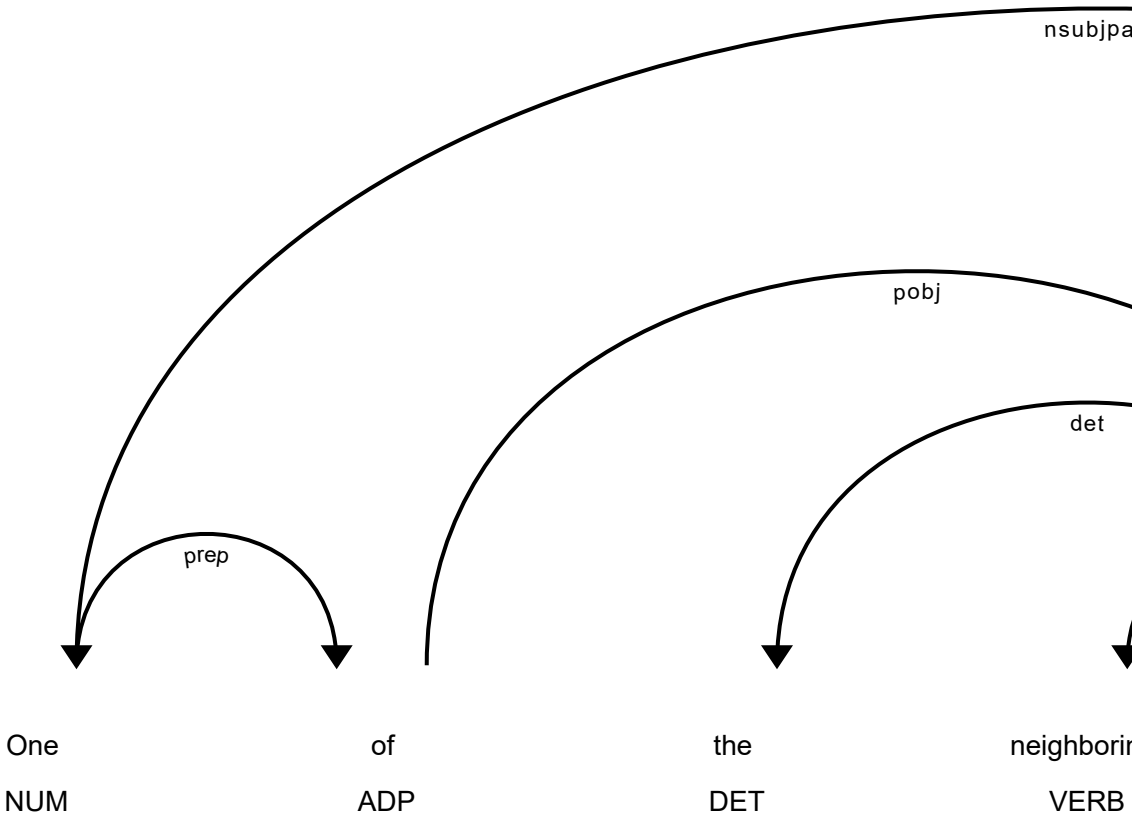
In [35]:

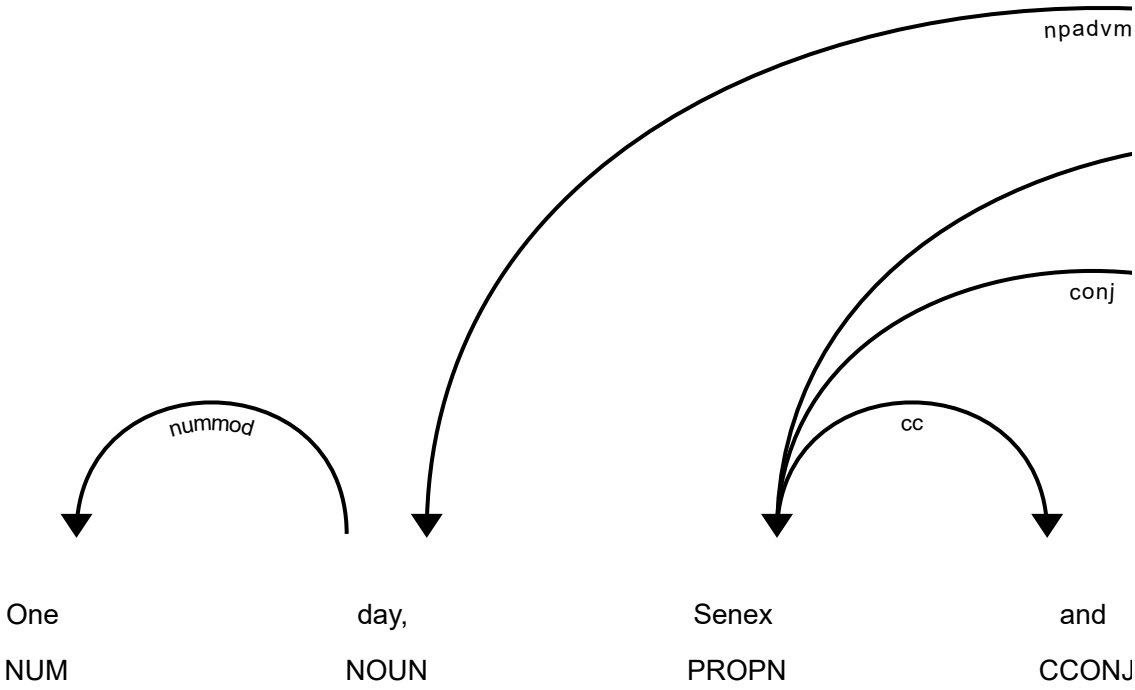
```
import spacy
from spacy import displacy

nlp = spacy.load("en_core_web_sm")
text = """In ancient Rome, some neighbors live in three adjacent houses. In the center is t
doc = nlp(text)
sentence_spans = list(doc.sents)
displacy.render(sentence_spans, style="dep")
```

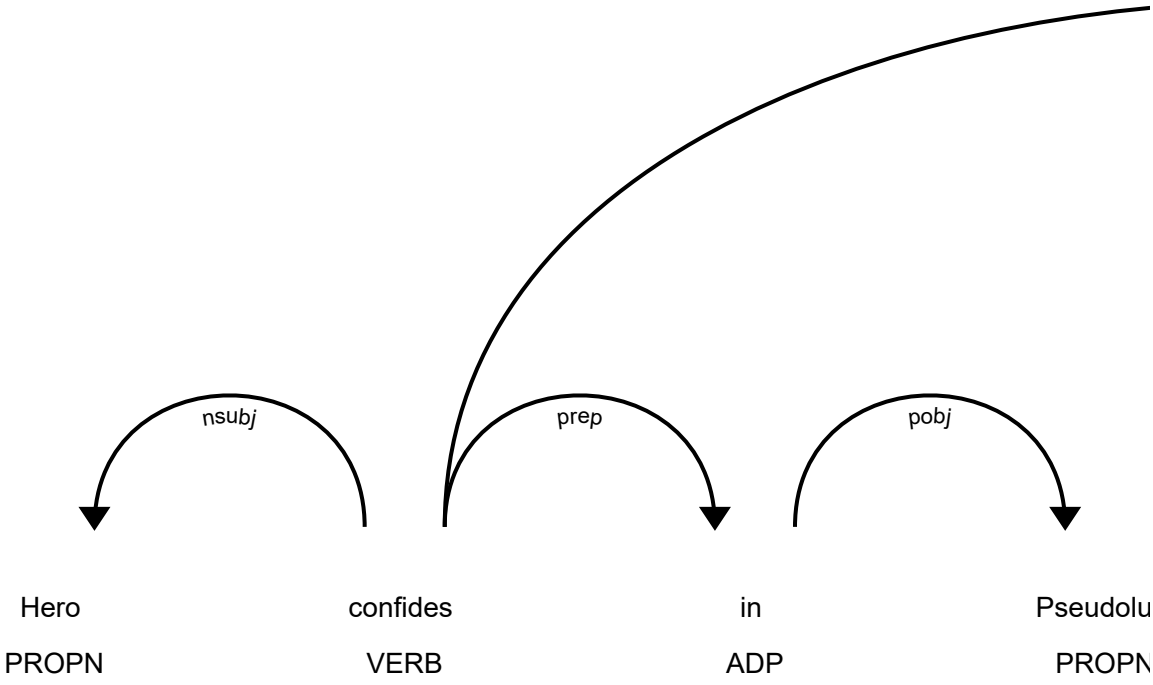



prep
pobj
det
In
ADP
the
DET
center
NOUN
is
AUX


det
acl
prep
A
DET
slave
NOUN
belonging
VERB
to
ADP


nsubjpa
pobj
det
prep
One
NUM
of
ADP
the
DET
neighborir
VERB


npadvm
conj
nummod
cc
One
NUM
day,
NOUN
Senex
PROPN
and
CCONJ


nsubj
prep
pobj
Hero
PROPN
confides
VERB
in
ADP
Pseudolu
PROPN

In [34]:

```
colors = {"ORG": "linear-gradient(90deg, #aa9cfc, #fc9ce7)"}
options = {"ents": ["ORG"], "colors": colors}
displacy.render(doc, style="ent", options=options)
```

When Sebastian Thrun started working on self-driving cars at Google ORG in 2007, few

people outside of the company took him seriously.

In [33]:

```
import spacy
from spacy import displacy

text = "When Sebastian Thrun started working on self-driving cars at Google in 2007, few pe

nlp = spacy.load("en_core_web_sm")
doc = nlp(text)
displacy.render(doc, style="ent")
```

When Sebastian Thrun PERSON started working on self-driving cars at Google ORG in

2007 DATE , few people outside of the company took him seriously.

Facts & Figures

Here's a quick comparison of the functionalities offered by spaCy, NLTK and CoreNLP.

	SPACY	NLTK	CORENLP
Programming language	Python	Python	Java / Python
Neural network models	✔	✘	✔
Integrated word vectors	✔	✘	✘
Multi-language support	✔	✔	✔
Tokenization	✔	✔	✔
Part-of-speech tagging	✔	✔	✔
Sentence segmentation	✔	✔	✔
Dependency parsing	✔	✘	✔
Entity recognition	✔	✔	✔
Entity linking	✔	✘	✘
Coreference resolution	✘	✘	✔

- Detailed speed comparison Here they compare the per-document processing time of various spaCy functionalities against other NLP libraries. They showed both absolute timings (in ms) and relative performance (normalized to spaCy). Lower is better.

	ABSOLUTE (MS PER DOC)			RELATIVE (TO SPACY)		
SYSTEM	TOKENIZE	TAG	PARSE	TOKENIZE	TAG	PARSE
spaCy	0.2ms	1ms	19ms	1x	1x	1x
CoreNLP	0.18ms	10ms	49ms	0.9x	10x	2.6x
ZPar	1ms	8ms	850ms	5x	8x	44.7x
NLTK	4ms	443ms	*n/a*	20x	443x	*n/a*

Vielen Dank

In [ ]: