Advance Preprocessing

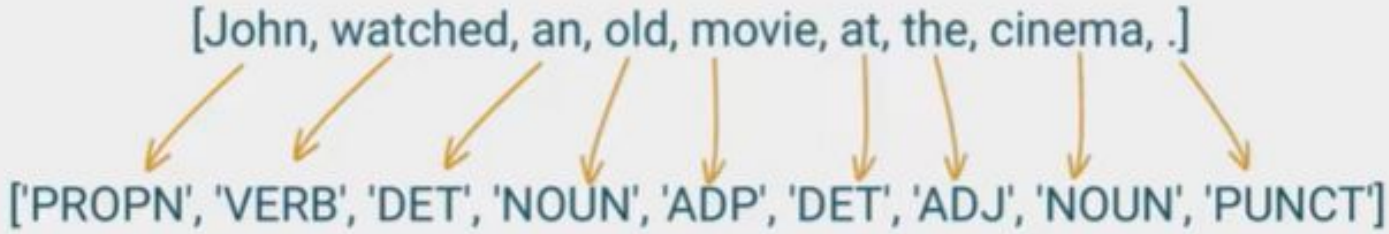
PRAT-OF-SPEECH TAGGING, NAMED ENTITY RECOGNITION AND PARSING

Part-of-Speech (POS) Tagging

Classifying how a word is used in a sentence.

{ noun, verb, adjective, ... }

POS tagging assigns each word in a sentence its corresponding POS.



Code implementation of POS Tagging

Part-of-Speech Tagging

POS tags can be accessed through the *pos_* attribute

```
[(t.text, t.pos_) for t in doc]
```

```
[('John', 'PROPN'),
 ('watched', 'VERB'),
 ('an', 'DET'),
 ('old', 'ADJ'),
 ('movie', 'NOUN'),
 ('at', 'ADP'),
 ('the', 'DET'),
 ('cinema', 'NOUN'),
 ('.', 'PUNCT')]
```

To get a description for a POS tag, we can use *spacy.explain*.

```
spacy.explain('PROPN')
```

```
[5] [(t.text, t.tag_) for t in doc]
```

```
[('John', 'NNP'),
 ('watched', 'VBD'),
 ('an', 'DT'),
 ('old', 'JJ'),
 ('movie', 'NN'),
 ('at', 'IN'),
 ('the', 'DT'),
 ('cinema', 'NN'),
 ('.', '.')]
```

So **NNP** refers specifically to a *singular pronoun*, and **VBD** is a verb in *past tense*.

```
print(spacy.explain('NNP'))
print(spacy.explain('VBD'))
```

```
noun, proper singular
verb, past tense
```

Named Entity Recognition (NER)

Tagging named ("real-world") entities.

{ a person, a location, an organization, ... }

Named Entity: (roughly) anything that can be referred by a proper name. They often have a *Proper Noun (PROPN)* POS tag.

Most Common

Person (PER)
"Alice", "Isaac Newton"

Location (LOC)
"Bay Area", "Rocky Mountains"

Geopolitical Entity (GPE)
"Canada", "Chicago"

Organization (ORG)
"Microsoft", "Porsche"

Why NER is useful?

Useful in a variety of tasks and applications...

Organizing/Categorizing corpus

e.g. identify medical procedures or diseases in research, or categorize support tickets based on entities mentioned.

Question answering

e.g. extract entities from a question and use NER to narrow down possible candidate answers. A question about a country's capital is going to result in an answer that's either LOC or GPE.

Critical in information extraction

e.g. extracting events and relationships between entities.

Challenges...

An entity can span multiple tokens

"Alexander Hamilton"

Entity *spans* two tokens and the system must identify the boundaries correctly.

Type ambiguity

Code implementation of NER Tagging

Named Entity Recognition

There are multiple ways to access named entities. One way is through the *ent_type_* attribute.

```
s = "Volkswagen is developing an electric sedan which could potentially come to America next fall."
doc = nlp(s)

[(t.text, t.ent_type_) for t in doc]
```

```
[('Volkswagen', 'ORG'),
 ('is', ''),
 ('developing', ''),
 ('an', ''),
 ('electric', ''),
 ('sedan', ''),
 ('which', ''),
 ('could', ''),
 ('potentially', ''),
 ('come', ''),
 ('to', ''),
 ('America', 'GPE'),
 ('next', 'DATE'),
 ('fall', 'DATE'),
 ('.', '')]
```

You can also check if a token is an entity before printing it by checking whether the *ent_type* (note the lack of trailing underscore) attribute is non-zero.

```
[9] print([(t.text, t.ent_type_) for t in doc if t.ent_type != 0])
```

```
[('Volkswagen', 'ORG'), ('America', 'GPE'), ('next', 'DATE'), ('fall', 'DATE')]
```

Another way is through the *ents* property of the **Doc** object. Here, we iterate through *ents* and print the entity itself and its label.

```
print([(ent.text, ent.label_) for ent in doc.ents])
```

+ Code + Text

Parsing

Determining the syntactic structure of a sentence.

Two common approaches

Constituency Parsing

Dependency Parsing

Constituency Parsing uses a *context-free grammar (CFG)*

A set of rules on how to build valid sentences starting from smaller building blocks.

e.g. for English, the grammar would specify how to assemble words into phrases and clauses, and how to further assemble phrases and clauses into sentences.

NP	Noun Phrase
VP	Verb Phrase
PP	Prepositional Phrase
NN	Noun
PRP	Personal Pronoun
NNP	Proper Noun
VB	Verb (base form)
DT	Determiner
IN	Preposition

CFG consists of two things:

Production Rules

S ⇒ NP VP

NP ⇒ PRP | NNP | DT NN

VP ⇒ VB | VB NP | VP PP

PP ⇒ IN NP

Lexicon

DT ⇒ the | a | this | that

PRP ⇒ I | she | he

IN ⇒ in | at

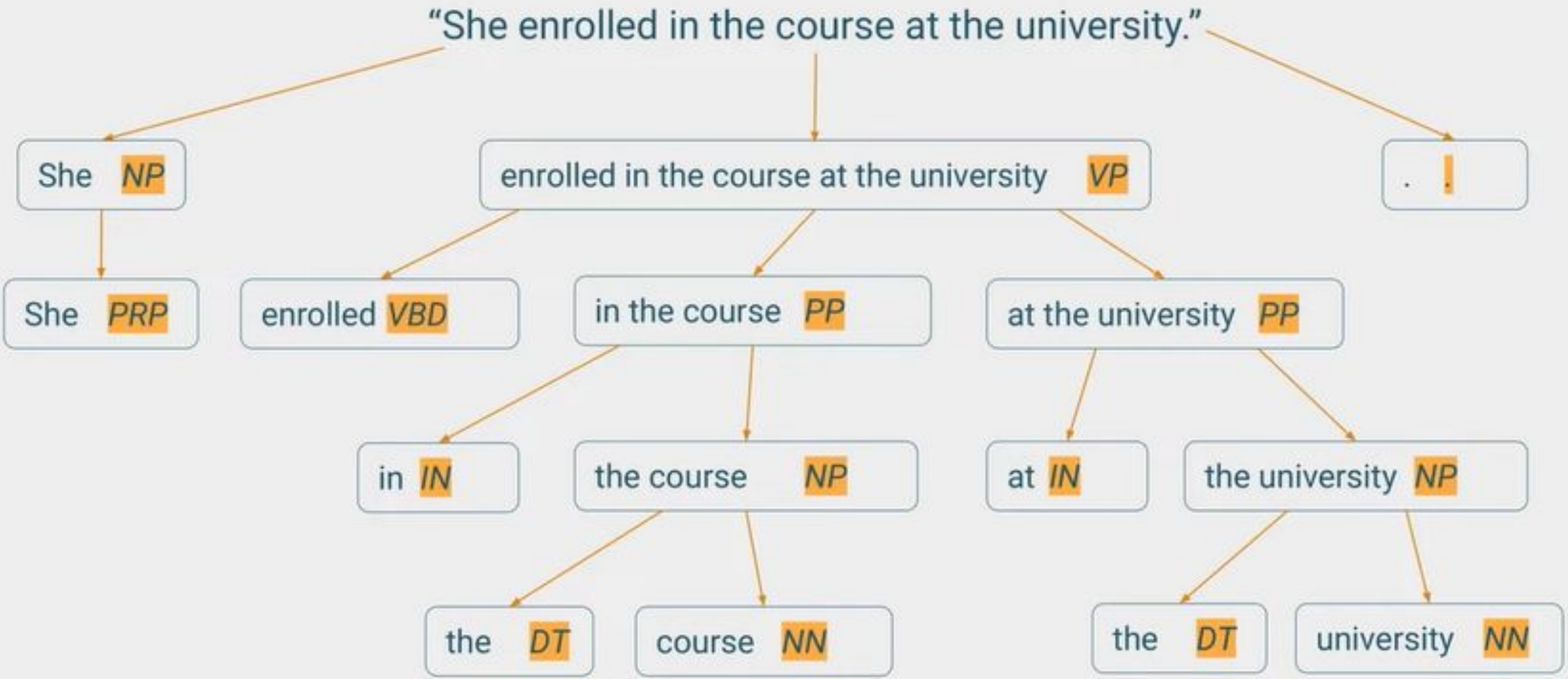
VB ⇒ book | fly | run

NN ⇒ book | hotel | room

"The room was booked"

DT NN

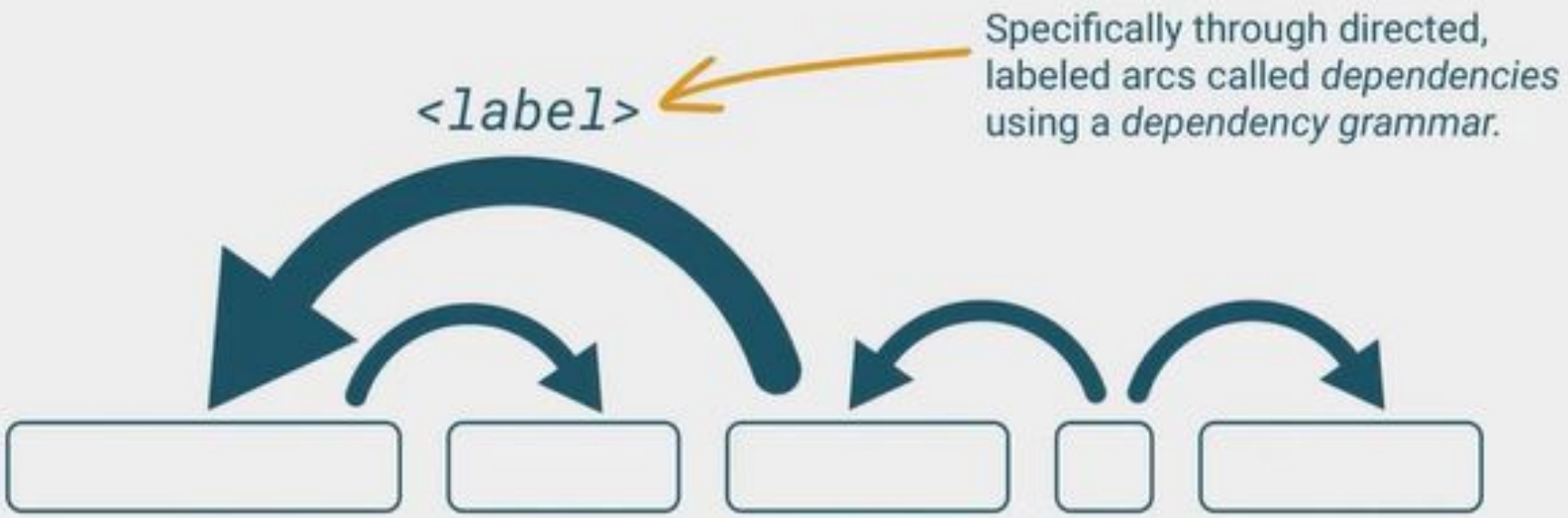
Constituency Parsing example



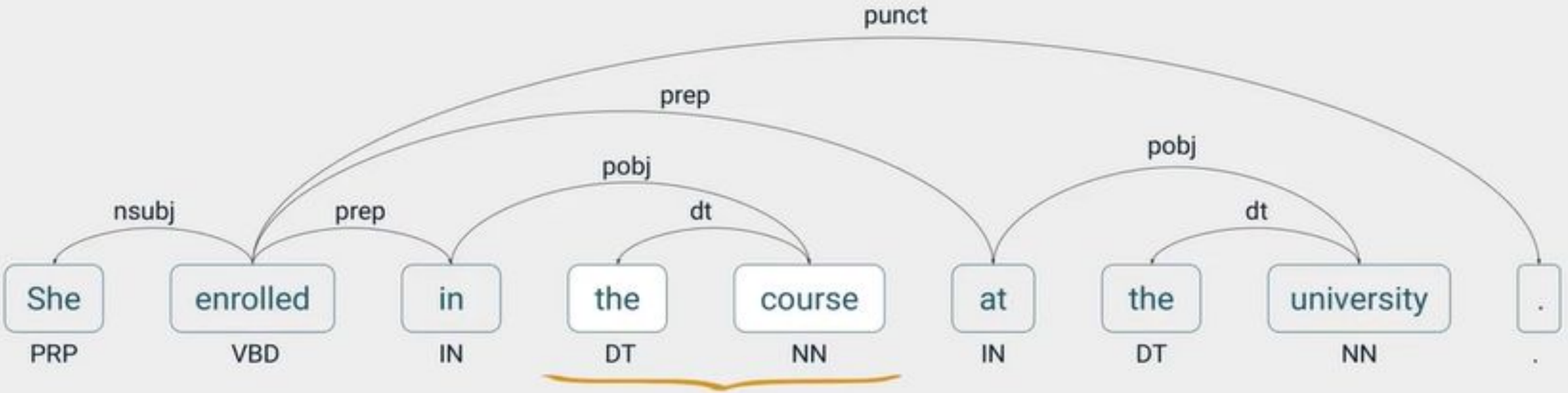
Parsing

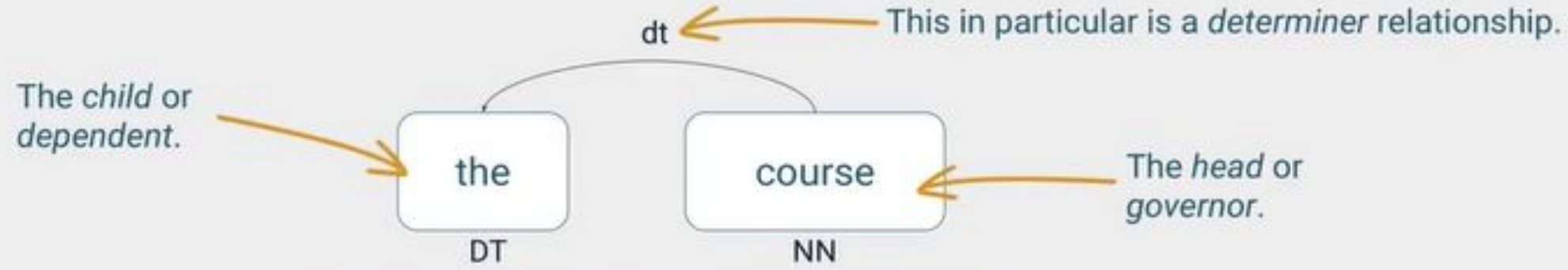
Determining the syntactic structure of a sentence.

Dependency Parsing describes binary relationships between words.



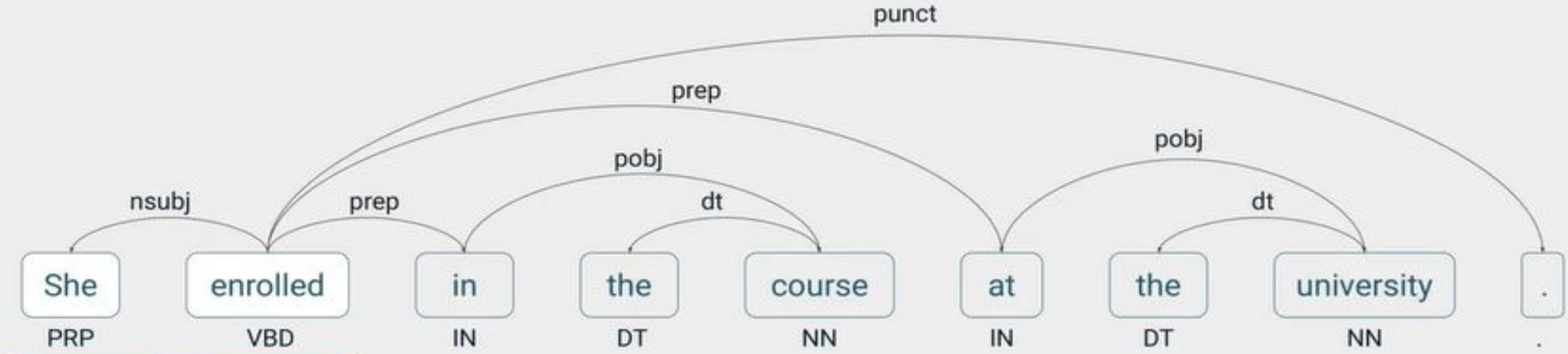
Dependency parse of "She enrolled in the course at the university."

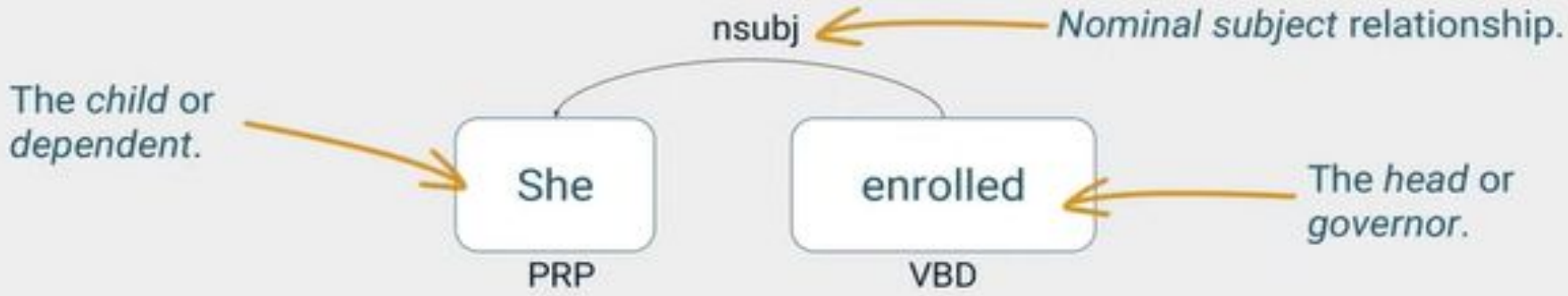




Here, we say that "the" *depends* on "course".

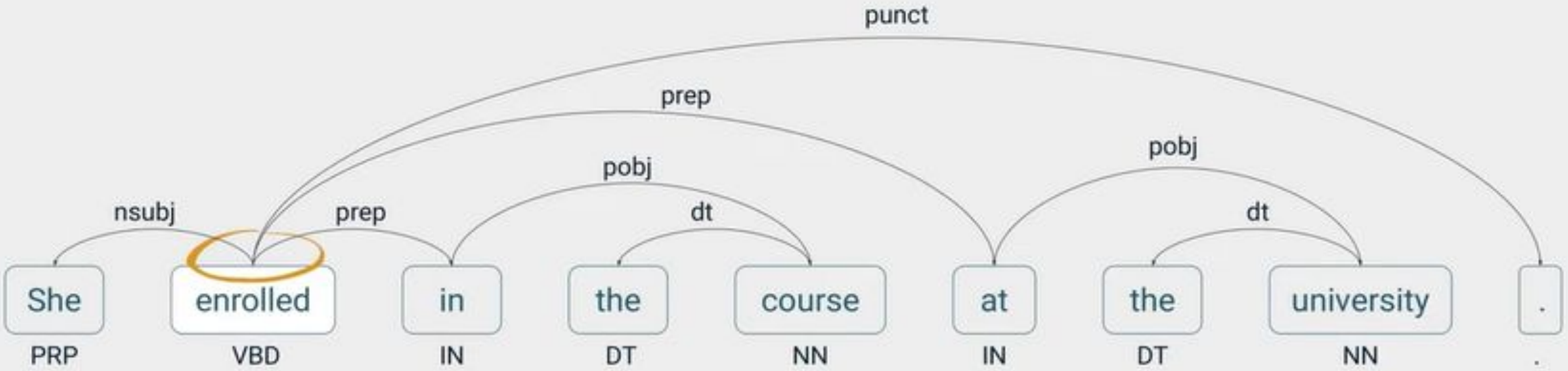
Dependency parse of "She enrolled in the course at the university."





"She" is the *nominal subject* of "enrolled". "She" *depends* on "enrolled".

Dependency parse of "She enrolled in the course at the university."



Notice the word "enrolled" has no arcs pointing to it. In this sentence, it acts as the *root*. The *finite verb* is often the root of a sentence.

A word can be a child to only **one** head, while the same word can act as a head to zero, one, or multiple words.

Constituency Parsing vs Dependency Parsing

Which one you use will depend on your goals.

In general: If you're interested in extracting sub-phrases from a sentence, use a constituency parse.

If your goal requires knowing the semantic relationship between words for applications such as question answering, it's easier to get them from a dependency parse. Dependency parsing is also useful for languages with more relaxed rules on word ordering.

Preprocessing RECAP

Perhaps your application requires custom steps.

Translating emojis to text labels.

Language detection in a mixed-language corpus.

Spelling correction or word fill-in for audio transcripts.

Thank You