Data Structures: Vocab, Lexemes and StringStore

ADVANCED NLP WITH SPACY

Ines Montani
spaCy core developer

DataCamp

Shared vocab and string store (1)

- `Vocab` : stores data shared across multiple documents
- To save memory, spaCy encodes all strings to **hash values**
- Strings are only stored once in the `StringStore` via `nlp.vocab.strings`
- String store: **lookup table** in both directions

```
coffee_hash = nlp.vocab.strings['coffee']
coffee_string = nlp.vocab.strings[coffee_hash]
```

  - Hashes can't be reversed – that's why we need to provide the shared vocab

```
# Raises an error if we haven't seen the string before
string = nlp.vocab.strings[3197928453018144401]
```

Shared vocab and string store (2)

- Look up the string and hash in `nlp.vocab.strings`

```
doc = nlp("I love coffee")
print('hash value:', nlp.vocab.strings['coffee'])
print('string value:', nlp.vocab.strings[3197928453018144401])
```

```
hash value: 3197928453018144401
string value: coffee
```

- The `doc` also exposes the vocab and strings

```
doc = nlp("I love coffee")
```

Lexemes: entries in the vocabulary

- A `Lexeme` object is an entry in the vocabulary

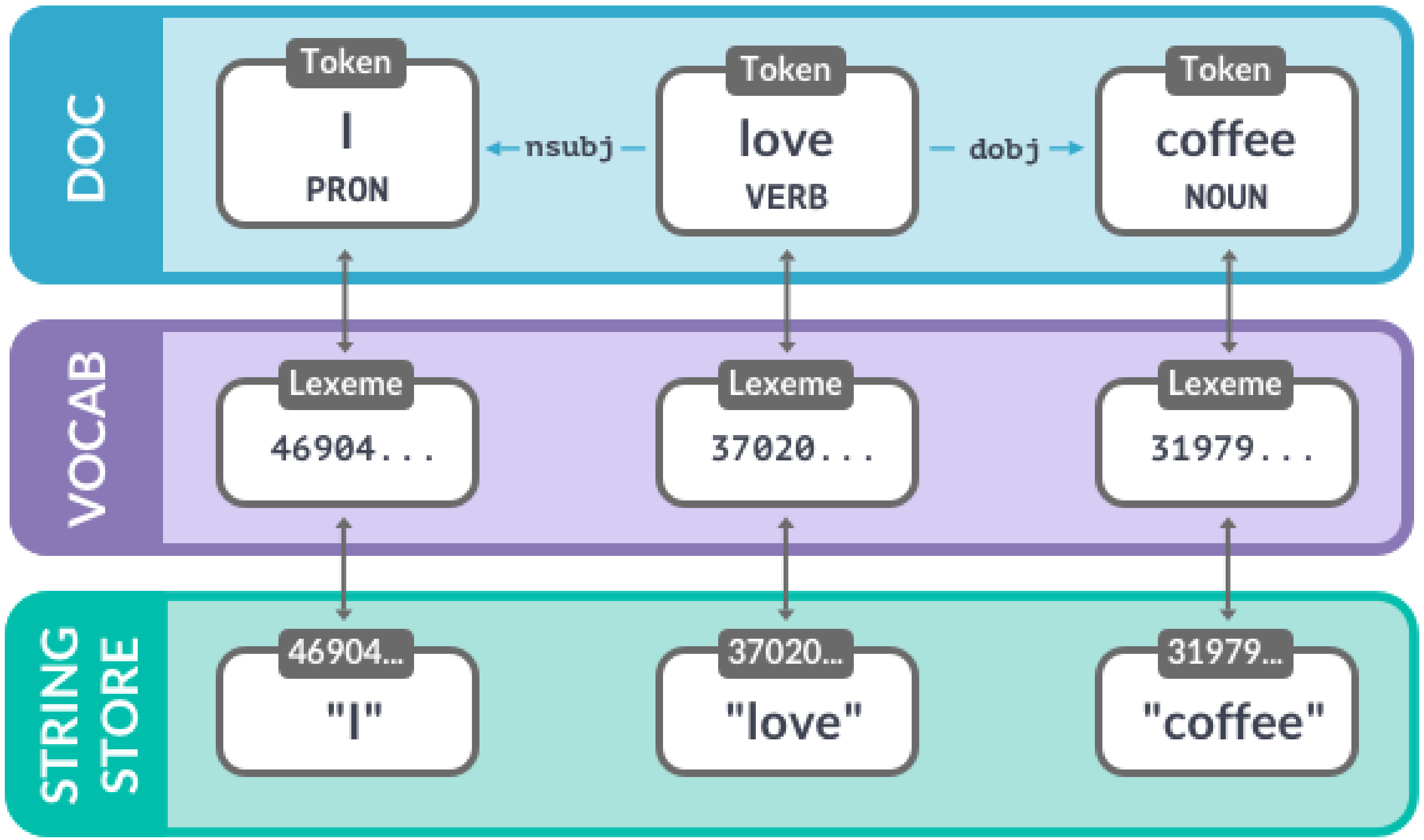
```python
doc = nlp("I love coffee")
lexeme = nlp.vocab['coffee']

# print the lexical attributes
print(lexeme.text, lexeme.orth, lexeme.is_alpha)
```

```
coffee 3197928453018144401 True
```

- Contains the **context-independent** information about a word
  - Word text: `lexeme.text` and `lexeme.orth` (the hash)
  - Lexical attributes like `lexeme.is_alpha`

Vocab, hashes and lexemes



Let's practice!

ADVANCED NLP WITH SPACY

DataCamp

Data Structures: Doc, Span and Token

ADVANCED NLP WITH SPACY

Ines Montani
spaCy core developer

DataCamp

The Doc object

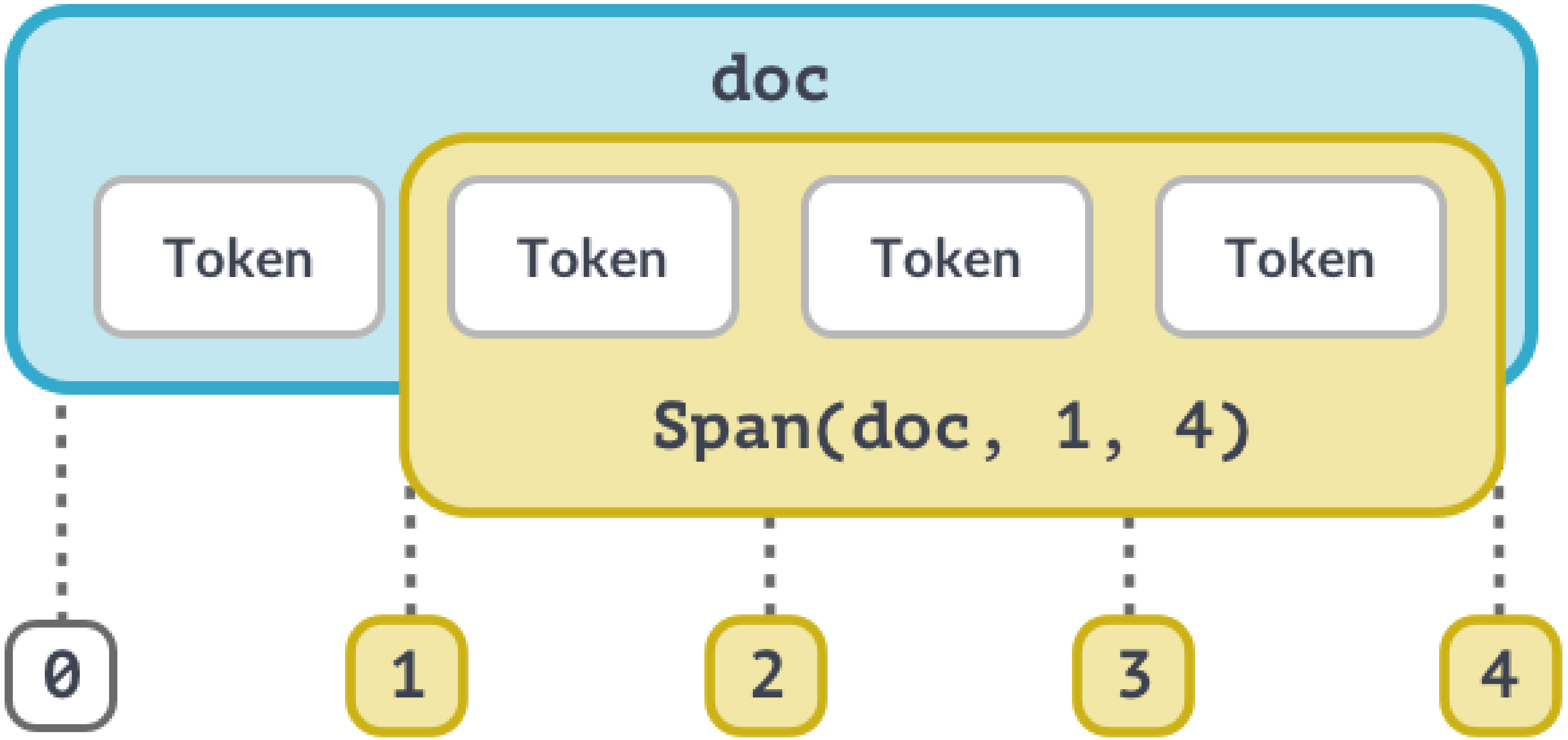
```
# Create an nlp object
from spacy.lang.en import English
nlp = English()

# Import the Doc class
from spacy.tokens import Doc

# The words and spaces to create the doc from
words = ['Hello', 'world', '!']
spaces = [True, False, False]

# Create a doc manually
doc = Doc(nlp.vocab, words=words, spaces=spaces)
```

The Span object (1)



The Span object (2)

```python
# Import the Doc and Span classes
from spacy.tokens import Doc, Span

# The words and spaces to create the doc from
words = ['Hello', 'world', '!']
spaces = [True, False, False]

# Create a doc manually
doc = Doc(nlp.vocab, words=words, spaces=spaces)

# Create a span manually
span = Span(doc, 0, 2)

# Create a span with a label
```

Best practices

- `Doc` and `Span` are very powerful and hold references and relationships of words and sentences
    - **Convert result to strings as late as possible**
    - **Use token attributes if available** – for example, `token.i` for the token index
- Don't forget to pass in the shared `vocab`

Let's practice!

ADVANCED NLP WITH SPACY

DataCamp

Word vectors and semantic similarity

ADVANCED NLP WITH SPACY

Ines Montani
spaCy core developer

DataCamp

Comparing semantic similarity

- `spaCy` can compare two objects and predict similarity
- `Doc.similarity()`, `Span.similarity()` and `Token.similarity()`
- Take another object and return a similarity score (`0` to `1`)
- **Important:** needs a model that has word vectors included, for example:
  - ? `en_core_web_md` (medium model)
  - ? `en_core_web_lg` (large model)
  - ???? **NOT** `en_core_web_sm` (small model)

Similarity examples (1)

```
# Load a larger model with vectors
nlp = spacy.load('en_core_web_md')

# Compare two documents
doc1 = nlp("I like fast food")
doc2 = nlp("I like pizza")
print(doc1.similarity(doc2))
```

```
0.8627204117787385
```

```
# Compare two tokens
doc = nlp("I like pizza and pasta")
```

Similarity examples (2)

```python
# Compare a document with a token
doc = nlp("I like pizza")
token = nlp("soap")[0]

print(doc.similarity(token))
```

```
0.32531983166759537
```

```python
# Compare a span with a document
span = nlp("I like pizza and pasta")[2:5]
doc = nlp("McDonalds sells burgers")
```

How does spaCy predict similarity?

- Similarity is determined using **word vectors**
- Multi-dimensional meaning representations of words
- Generated using an algorithm like **Word2Vec** and lots of text
- Can be added to spaCy's statistical models
- Default: cosine similarity, but can be adjusted
- `Doc` and `Span` vectors default to average of token vectors
- Short phrases are better than long documents with many irrelevant words

Word vectors in spaCy

```python
# Load a larger model with vectors
nlp = spacy.load('en_core_web_md')

doc = nlp("I have a banana")
# Access the vector via the token.vector attribute
print(doc[3].vector)
```

```
[2.02280000e-01,  -7.66180009e-02,   3.70319992e-01,
 3.28450017e-02,  -4.19569999e-01,   7.20689967e-02,
-3.74760002e-01,   5.74599989e-02,  -1.24009997e-02,
 5.29489994e-01,  -5.23800015e-01,  -1.97710007e-01,
-3.41470003e-01,   5.33169985e-01,  -2.53309999e-02,
```

Similarity depends on the application context

- Useful for many applications: recommendation systems, flagging duplicates etc.
- There's no objective definition of "similarity"
- Depends on the context and what application needs to do

```python
doc1 = nlp("I like cats")
doc2 = nlp("I hate cats")

print(doc1.similarity(doc2))
```

```
0.9501447503553421
```

Let's practice!

ADVANCED NLP WITH SPACY

DataCamp

Combining models and rules

ADVANCED NLP WITH SPACY

Ines Montani
spaCy core developer

DataCamp

Statistical predictions vs. rules

	Statistical models	Rule-based systems
Use cases	application needs to *generalize* based on examples	???????????? ???????????? ???????
Real-world examples	product names, person names, subject/object relationships	
spaCy features	entity recognizer, dependency parser, part-of-speech tagger	

Statistical predictions vs. rules

	Statistical models	Rule-based systems
Use cases	application needs to *generalize* based on examples	dictionary with finite number of examples
Real-world examples	product names, person names, subject/object relationships	countries of the world, cities, drug names, dog breeds
spaCy features	entity recognizer, dependency parser, part-of-speech tagger	tokenizer, `Matcher` , `PhraseMatcher`

Recap: Rule-based Matching

```
# Initialize with the shared vocab
from spacy.matcher import Matcher
matcher = Matcher(nlp.vocab)

# Patterns are lists of dictionaries describing the tokens
pattern = [{'LEMMA': 'love', 'POS': 'VERB'}, {'LOWER': 'cats'}]
matcher.add('LOVE_CATS', None, pattern)

# Operators can specify how often a token should be matched
pattern = [{'TEXT': 'very', 'OP': '+'}, {'TEXT': 'happy'}]

# Calling matcher on doc returns list of (match_id, start, end) tuples
doc = nlp("I love cats and I'm very very happy")
matches = matcher(doc)
```

Adding statistical predictions

```
matcher = Matcher(nlp.vocab)
matcher.add('DOG', None, [{'LOWER': 'golden'}, {'LOWER': 'retriever'}])
doc = nlp("I have a Golden Retriever")

for match_id, start, end in matcher(doc):
    span = doc[start:end]
    print('Matched span:', span.text)

    # Get the span's root token and root head token
    print('Root token:', span.root.text)
    print('Root head token:', span.root.head.text)

    # Get the previous token and its POS tag
    print('Previous token:', doc[start - 1].text, doc[start - 1].pos_)
```

Efficient phrase matching (1)

- `PhraseMatcher` like regular expressions or keyword search – but with access to the tokens!
- Takes `Doc` object as patterns
- More efficient and faster than the `Matcher`
- Great for matching large word lists

Efficient phrase matching (2)

```
from spacy.matcher import PhraseMatcher


matcher = PhraseMatcher(nlp.vocab)

pattern = nlp("Golden Retriever")
matcher.add('DOG', None, pattern)


doc = nlp("I have a Golden Retriever")

# iterate over the matches
for match_id, start, end in matcher(doc):
    # get the matched span
    span = doc[start:end]
    print('Matched span:', span.text)
```

Let's practice!

ADVANCED NLP WITH SPACY

DataCamp