

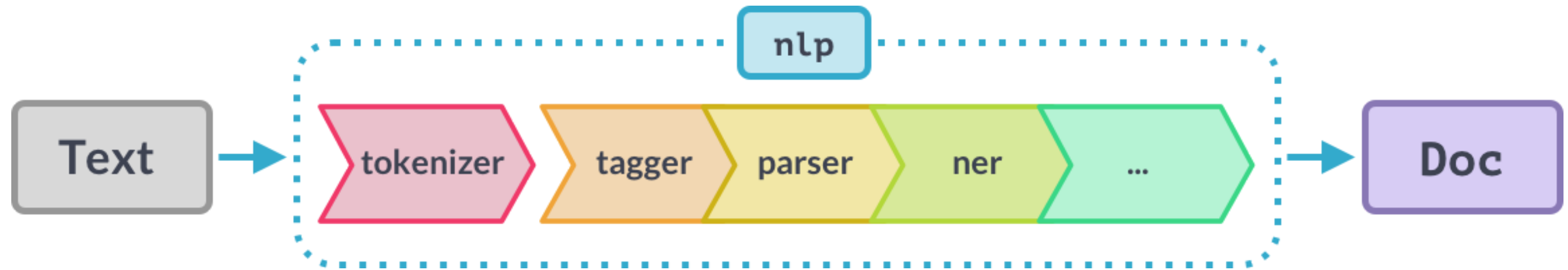
# Processing pipelines

ADVANCED NLP WITH SPACY



**Ines Montani**  
spaCy core developer

# What happens when you call nlp?

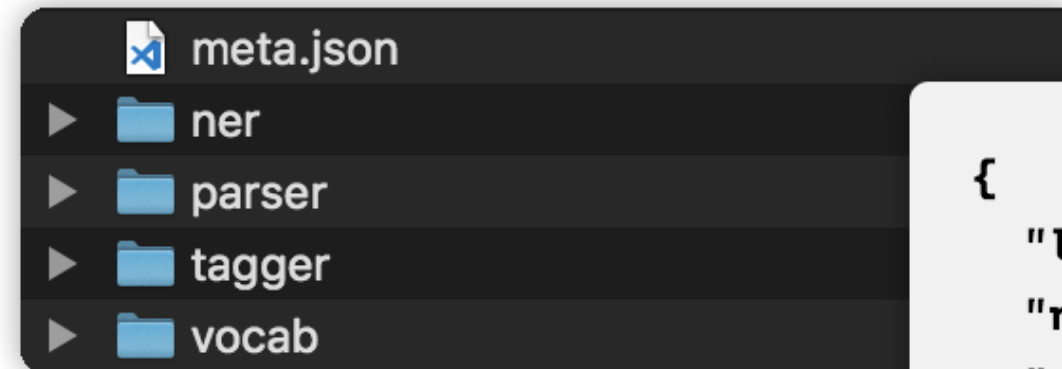


```
doc = nlp("This is a sentence.")
```

# Built-in pipeline components

Name	Description	Creates
<b>tagger</b>	Part-of-speech tagger	<code>Token.tag</code>
<b>parser</b>	Dependency parser	<code>Token.dep</code> , <code>Token.head</code> , <code>Doc.sents</code> , <code>Doc.noun_chunks</code>
<b>ner</b>	Named entity recognizer	<code>Doc.ents</code> , <code>Token.ent_iob</code> , <code>Token.ent_type</code>
<b>textcat</b>	Text classifier	<code>Doc.cats</code>

# Under the hood



meta.json

```
{  
  "lang": "en",  
  "name": "core_web_sm",  
  "pipeline": ["tagger", "parser", "ner"]  
}
```

- Pipeline defined in model's `meta.json` in order
- Built-in components need binary data to make predictions

# Pipeline attributes

- `nlp.pipe_names` : list of pipeline component names

```
print(nlp.pipe_names)
```

```
['tagger', 'parser', 'ner']
```

- `nlp.pipeline` : list of `(name, component)` tuples

```
print(nlp.pipeline)
```

```
[('tagger', <spacy.pipeline.Tagger>),  
 ('parser', <spacy.pipeline.DependencyParser>),  
 ('ner', <spacy.pipeline.EntityRecognizer>)]
```

# Let's practice!

ADVANCED NLP WITH SPACY

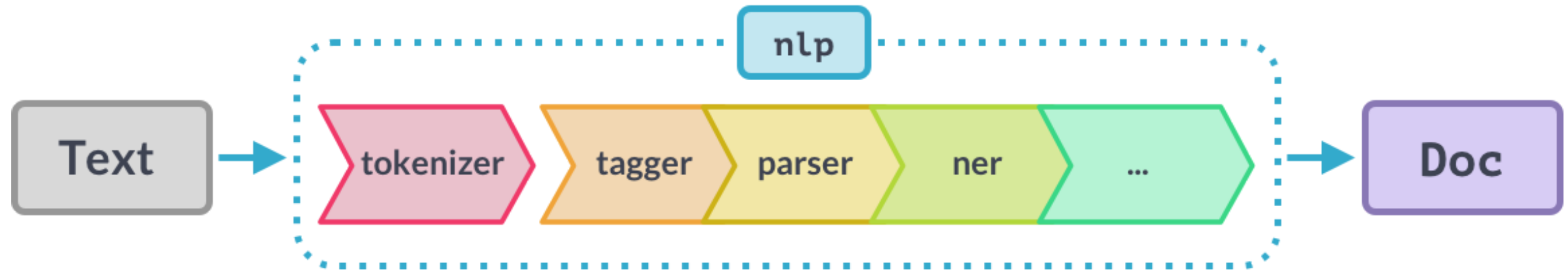
# Custom pipeline components

ADVANCED NLP WITH SPACY



Ines Montani  
spaCy core developer

# Why custom components?



- Make a function execute automatically when you call `nlp`
- Add your own metadata to documents and tokens
- Updating built-in attributes like `doc.ents`



# Anatomy of a component (1)

- Function that takes a `doc` , modifies it and returns it
- Can be added using the `nlp.add_pipe` method

```
def custom_component(doc):  
    # Do something to the doc here  
    return doc  
  
nlp.add_pipe(custom_component)
```

# Anatomy of a component (2)

```
def custom_component(doc):  
    # Do something to the doc here  
    return doc  
  
nlp.add_pipe(custom_component)
```

Argument	Description	Example
last	If True ,add last	nlp.add_pipe(component, last=True)
first	If True ,add first	nlp.add_pipe(component, first=True)
before	Add before component	nlp.add_pipe(component, before='ner')
after	Add after component	nlp.add_pipe(component, after='tagger')

# Example: a simple component (1)

```
# Create the nlp object
nlp = spacy.load('en_core_web_sm')

# Define a custom component
def custom_component(doc):
    # Print the doc's length
    print('Doc length:' len(doc))

    # Return the doc object
    return doc

# Add the component first in the pipeline
nlp.add_pipe(custom_component, first=True)

# Print the pipeline component names
print('Pipeline:', nlp.pipe_names)
```

```
Pipeline: ['custom_component', 'tagger', 'parser', 'ner']
```

# Example: a simple component (2)

```
# Create the nlp object
nlp = spacy.load('en_core_web_sm')

# Define a custom component
def custom_component(doc):

    # Print the doc's length
    print('Doc length:' len(doc))

    # Return the doc object
    return doc

# Add the component first in the pipeline
nlp.add_pipe(custom_component, first=True)

# Process a text
doc = nlp("Hello world!")
```

```
Doc length: 3
```

# Let's practice!

ADVANCED NLP WITH SPACY

# Extension attributes

ADVANCED NLP WITH SPACY



**Ines Montani**  
spaCy core developer

# Setting custom attributes

- Add custom metadata to documents, tokens and spans
- Accessible via the `._` property

```
doc._.title = 'My document'  
token._.is_color = True  
span._.has_color = False
```

- registered on the global `Doc` , `Token` or `Span` using the `set_extension` method

```
# Import global classes  
from spacy.tokens import Doc, Token, Span  
  
# Set extensions on the Doc, Token and Span  
Doc.set_extension('title', default=None)  
Token.set_extension('is_color', default=False)  
Span.set_extension('has_color', default=False)
```

# Extension attribute types

1. Attribute extensions
2. Property extensions
3. Method extensions



# Attribute extensions

- Set a default value that can be overwritten

```
from spacy.tokens import Token

# Set extension on the Token with default value
Token.set_extension('is_color', default=False)

doc = nlp("The sky is blue.")

# Overwrite extension attribute value
doc[3]._.is_color = True
```

# Property extensions (1)

- Define a getter and an optional setter function
- Getter only called when you *retrieve* the attribute value

```
from spacy.tokens import Token

# Define getter function
def get_is_color(token):
    colors = ['red', 'yellow', 'blue']
    return token.text in colors

# Set extension on the Token with getter
Token.set_extension('is_color', getter=get_is_color)

doc = nlp("The sky is blue.")
print(doc[3]._.is_color, '-', doc[3].text)
```

```
blue - True
```

# Property extensions (2)

- `Span` extensions should almost always use a getter

```
from spacy.tokens import Span

# Define getter function
def get_has_color(span):
    colors = ['red', 'yellow', 'blue']
    return any(token.text in colors for token in span)

# Set extension on the Span with getter
Span.set_extension('has_color', getter=get_has_color)

doc = nlp("The sky is blue.")
print(doc[1:4]._.has_color, '-', doc[1:4].text)
print(doc[0:2]._.has_color, '-', doc[0:2].text)
```

True - sky is blue

False - The sky

# Method extensions

- Assign a **function** that becomes available as an object method
- Lets you pass **arguments** to the extension function

```
from spacy.tokens import Doc

# Define method with arguments
def has_token(doc, token_text):
    in_doc = token_text in [token.text for token in doc]

# Set extension on the Doc with method
Doc.set_extension('has_token', method=has_token)

doc = nlp("The sky is blue.")
print(doc._.has_token('blue'), '- blue')
print(doc._.has_token('cloud'), '- cloud')
```

```
True - blue
False - cloud
```

# Let's practice!

ADVANCED NLP WITH SPACY

# Scaling and performance

ADVANCED NLP WITH SPACY



**Ines Montani**  
spaCy core developer

# Processing large volumes of text

- Use `nlp.pipe` method
- Processes texts as a stream, yields `Doc` objects
- Much faster than calling `nlp` on each text

**BAD:**

```
docs = [nlp(text) for text in LOTS_OF_TEXTS]
```

**GOOD:**

```
docs = list(nlp.pipe(LOTS_OF_TEXTS))
```

# Passing in context (1)

- Setting `as_tuples=True` on `nlp.pipe` lets you pass in `(text, context)` tuples
- Yields `(doc, context)` tuples
- Useful for associating metadata with the `doc`

```
data = [  
    ('This is a text', {'id': 1, 'page_number': 15}),  
    ('And another text', {'id': 2, 'page_number': 16}),  
]  
  
for doc, context in nlp.pipe(data, as_tuples=True):  
    print(doc.text, context['page_number'])
```

```
This is a text 15  
And another text 16
```



# Passing in context (2)

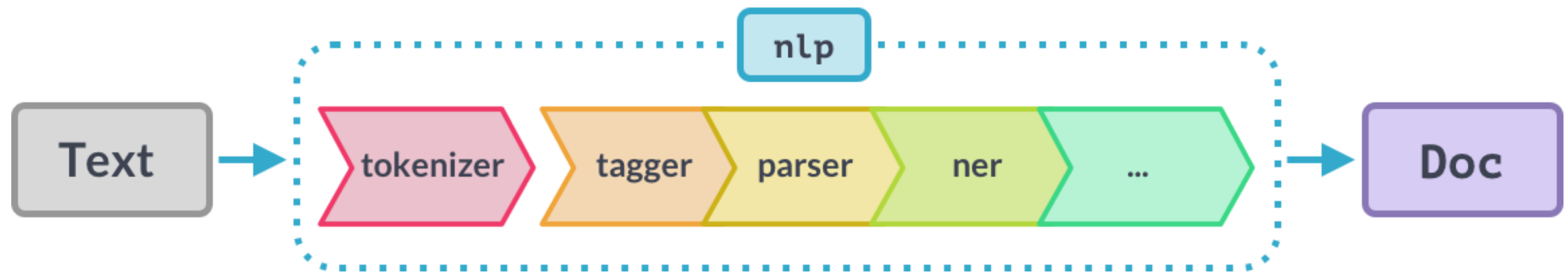
```
from spacy.tokens import Doc

Doc.set_extension('id', default=None)
Doc.set_extension('page_number', default=None)

data = [
    ('This is a text', {'id': 1, 'page_number': 15}),
    ('And another text', {'id': 2, 'page_number': 16}),
]

for doc, context in nlp.pipe(data, as_tuples=True):
    doc._.id = context['id']
    doc._.page_number = context['page_number']
```

# Using only the tokenizer



- don't run the whole pipeline!

# Using only the tokenizer (2)

- Use `nlp.make_doc` to turn a text in to a `Doc` object

**BAD:**

```
doc = nlp("Hello world")
```

**GOOD:**

```
doc = nlp.make_doc("Hello world!")
```

# Disabling pipeline components

- Use `nlp.disable_pipes` to temporarily disable one or more pipes

```
# Disable tagger and parser
with nlp.disable_pipes('tagger', 'parser'):
    # Process the text and print the entities
    doc = nlp(text)
    print(doc.ents)
```

- restores them after the `with` block
- only runs the remaining components

# Let's practice!

ADVANCED NLP WITH SPACY