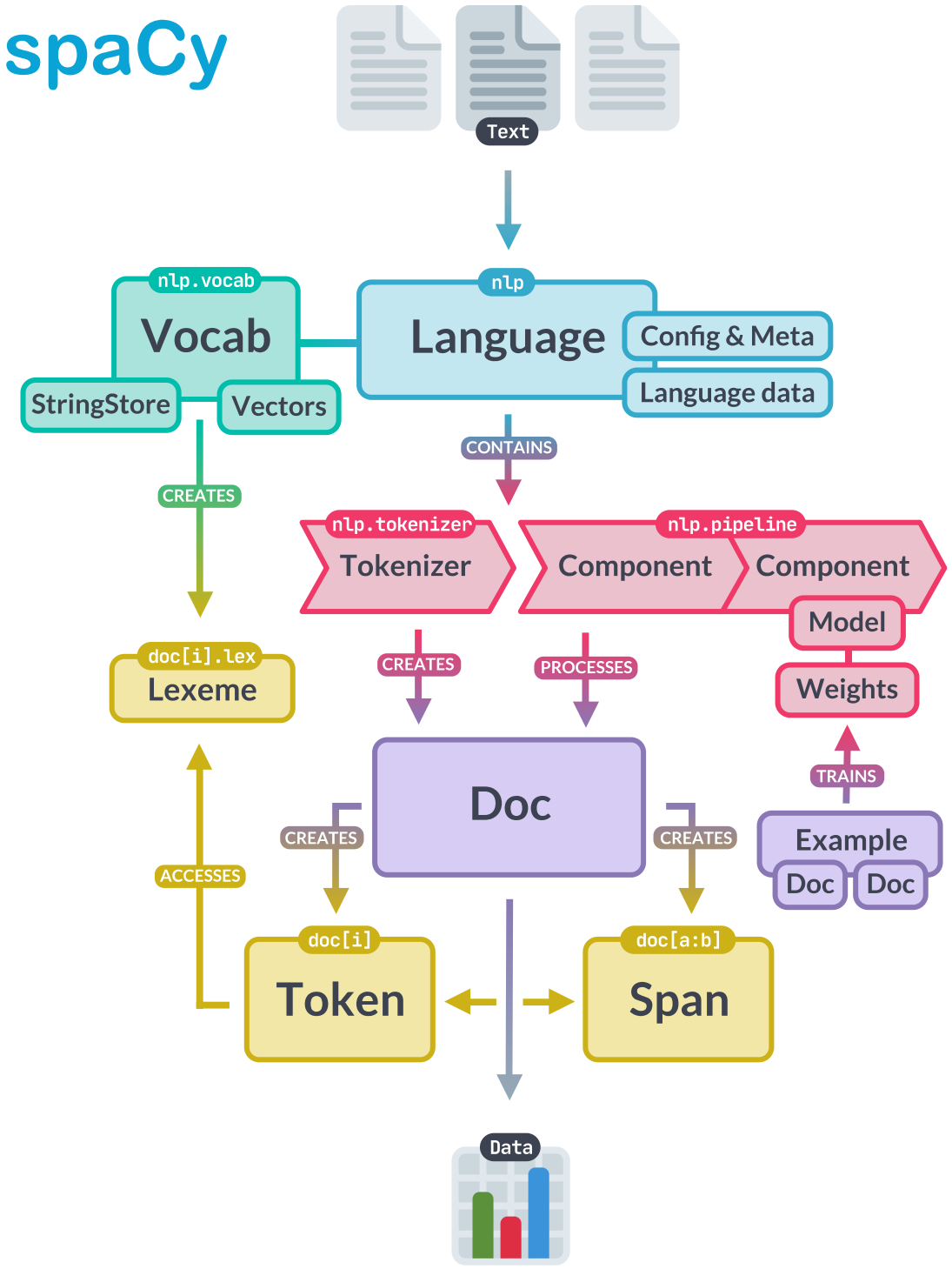
Library Architecture

The central data structures in spaCy are the `Language` class, the `Vocab` and the `Doc` object. The `Language` class is used to process a text and turn it into a `Doc` object. It's typically stored as a variable called `nlp`. The `Doc` object owns the **sequence of tokens** and all their annotations. By centralizing strings, word vectors and lexical attributes in the `Vocab`, we avoid storing multiple copies of this data. This saves memory, and ensures there's a **single source of truth**.

Text annotations are also designed to allow a single source of truth: the `Doc` object owns the data, and `Span` and `Token` are **views that point into it**. The `Doc` object is constructed by the `Tokenizer`, and then **modified in place** by the components of the pipeline. The `Language` object coordinates these components. It takes raw text and sends it through the pipeline, returning an **annotated document**. It also orchestrates training and serialization.

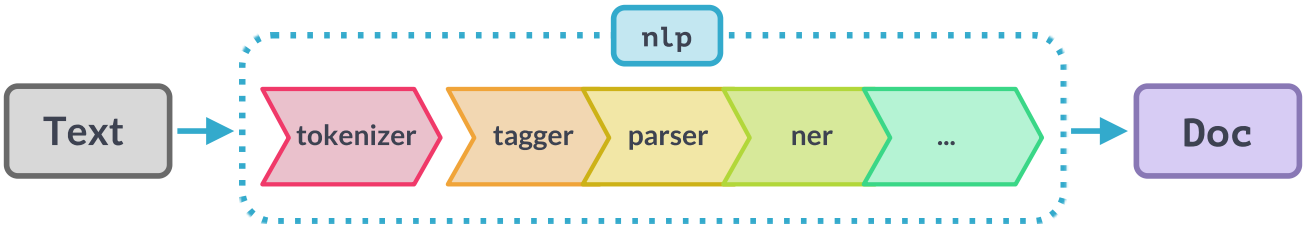


Container objects

NAME	DESCRIPTION
`Doc`	A container for accessing linguistic annotations.
`DocBin`	A collection of `Doc` objects for efficient binary serialization. Also used for training data .
`Example`	A collection of training annotations, containing two `Doc` objects: the reference data and the predictions.
`Language`	Processing class that turns text into `Doc` objects. Different languages implement their own subclasses of it. The variable is typically called `nlp` .
`Lexeme`	An entry in the vocabulary. It's a word type with no context, as opposed to a word token. It therefore has no part-of-speech tag, dependency parse etc.
`Span`	A slice from a `Doc` object.
`SpanGroup`	A named collection of spans belonging to a `Doc` .
`Token`	An individual token — i.e. a word, punctuation symbol, whitespace, etc.

Processing pipeline

The processing pipeline consists of one or more **pipeline components** that are called on the `Doc` in order. The tokenizer runs before the components. Pipeline components can be added using `Language.add_pipe` . They can contain a statistical model and trained weights, or only make rule-based modifications to the `Doc` . spaCy provides a range of built-in components for different language processing tasks and also allows adding custom components.



NAME	DESCRIPTION
`AttributeRuler`	Set token attributes using matcher rules.
`DependencyParser`	Predict syntactic dependencies.
`EditTreeLemmatizer`	Predict base forms of words.
`EntityLinker`	Disambiguate named entities to nodes in a knowledge base.
`EntityRecognizer`	Predict named entities, e.g. persons or products.
`EntityRuler`	Add entity spans to the `Doc` using token-based rules or exact phrase matches.
`Lemmatizer`	Determine the base forms of words using rules and lookups.
`Morphologizer`	Predict morphological features and coarse-grained part-of-speech tags.
`SentenceRecognizer`	Predict sentence boundaries.
`Sentencizer`	Implement rule-based sentence boundary detection that doesn't require the dependency parse.
`Tagger`	Predict part-of-speech tags.
`TextCategorizer`	Predict categories or labels over the whole document.
`Tok2Vec`	Apply a "token-to-vector" model and set its outputs.
`Tokenizer`	Segment raw text and create `Doc` objects from the words.
`TrainablePipe`	Class that all trainable pipeline components inherit from.
`Transformer`	Use a transformer model and set its outputs.
Other functions	Automatically apply something to the `Doc`, e.g. to merge spans of tokens.

Matchers

Matchers help you find and extract information from `Doc` objects based on match patterns describing the sequences you're looking for. A matcher operates on a `Doc` and gives you access to the matched tokens **in context**.

NAME	DESCRIPTION
`DependencyMatcher`	Match sequences of tokens based on dependency trees using Semgrex operators.
`Matcher`	Match sequences of tokens, based on pattern rules, similar to regular expressions.
`PhraseMatcher`	Match sequences of tokens based on phrases.

Other classes

NAME	DESCRIPTION
`Corpus`	Class for managing annotated corpora for training and evaluation data.
`KnowledgeBase`	Abstract base class for storage and retrieval of data for entity linking.
`InMemoryLookupKB`	Implementation of `KnowledgeBase` storing all data in memory.
`Candidate`	Object associating a textual mention with a specific entity contained in a `KnowledgeBase`.
`Lookups`	Container for convenient access to large lookup tables and dictionaries.
`MorphAnalysis`	A morphological analysis.
`Morphology`	Store morphological analyses and map them to and from hash values.
`Scorer`	Compute evaluation scores.
`StringStore`	Map strings to and from hash values.
`Vectors`	Container class for vector data keyed by string.
`Vocab`	The shared vocabulary that stores strings and gives you access to `Lexeme` objects.

SUGGEST EDITS