Natural Language Processing

The arc-standard algorithm

Marco Kuhlmann
Department of Computer and Information Science

The arc-standard algorithm

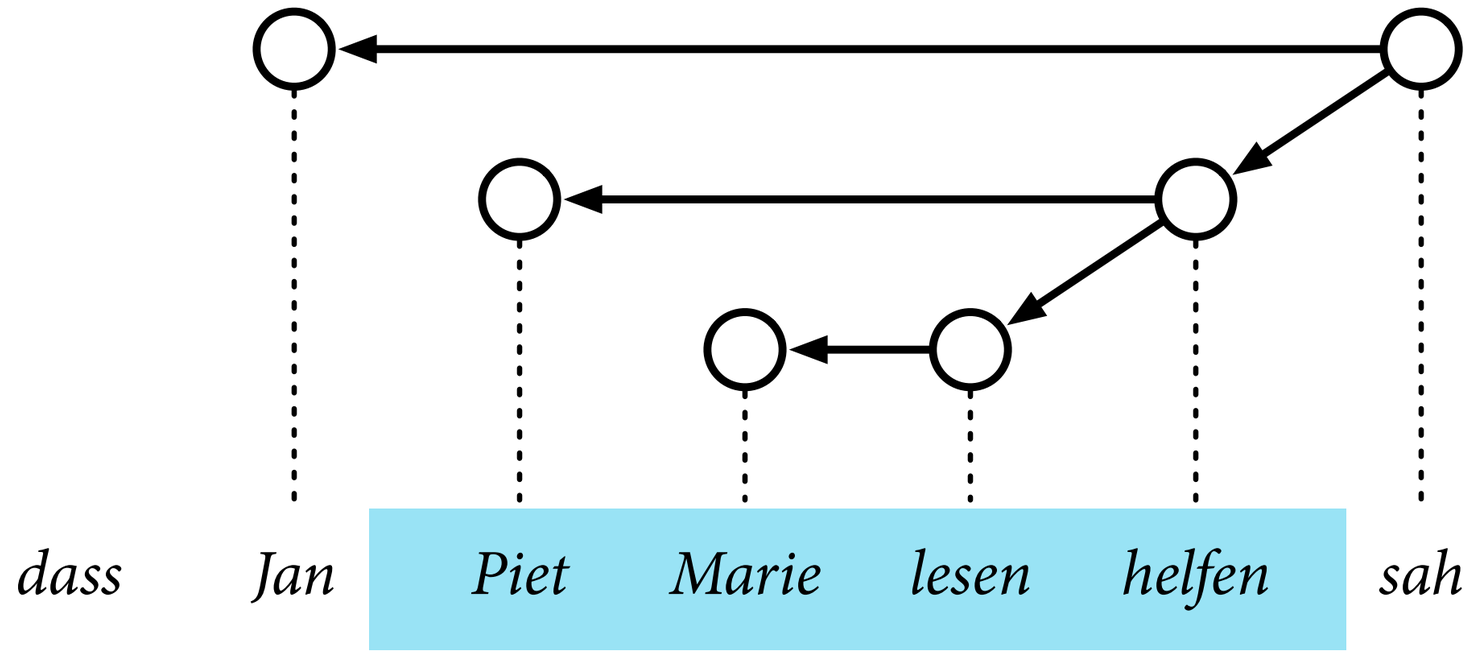
- The **arc-standard algorithm** is an algorithm for transition-based dependency parsing.
- It can be viewed as a generalisation of the shift–reduce algorithm for parsing context-free grammars.

  two types of “reduce” actions

- The arc-standard algorithm can only predict projective dependency trees.

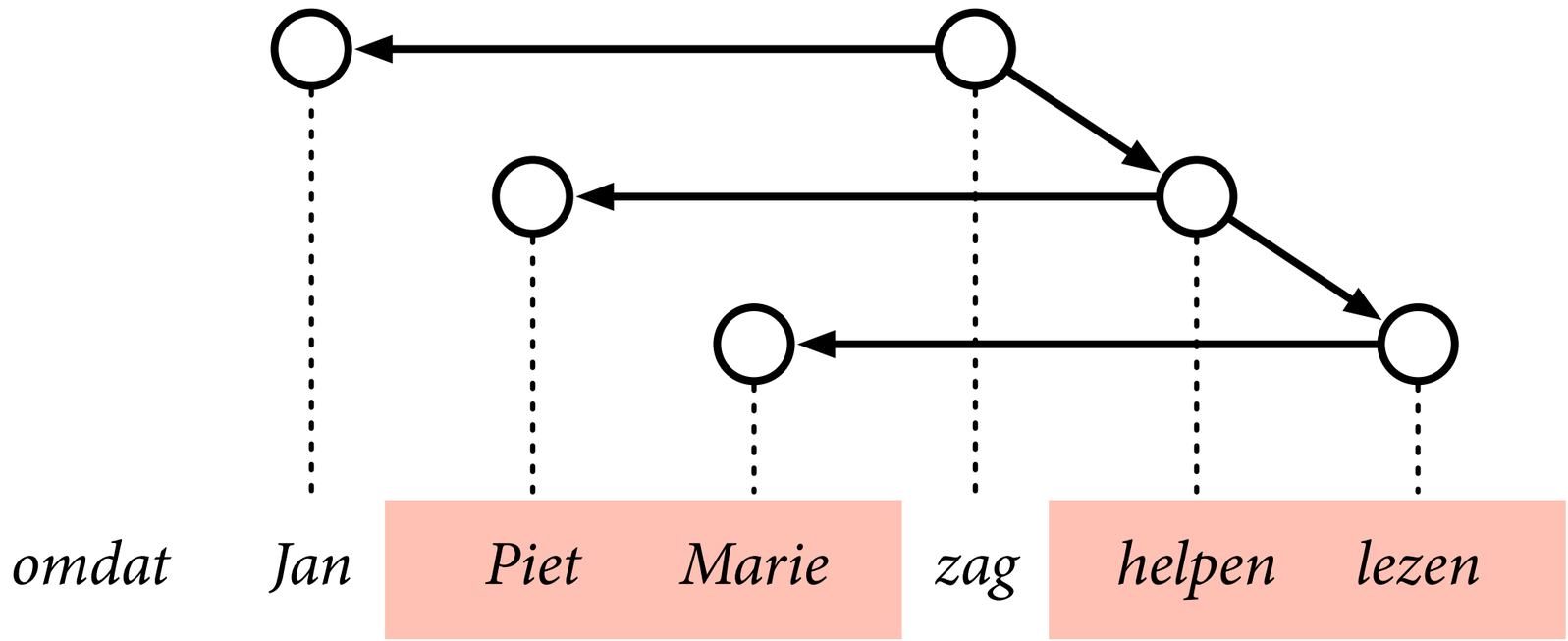
  Algorithms for non-projective trees exist, see e.g. Nivre (2009).

Projective dependency trees



Every subtree corresponds to a contiguous sequence of words.

Non-projective dependency trees



The sequence of words in a subtree may contain “gaps”.

Transition-based dependency parsing

- The parser starts in the **initial configuration**.

  empty dependency tree

- It then calls a classifier, which predicts the **transition** that the parser should make to move to a next configuration.

  extend the partial dependency tree

- This process is repeated until the parser reaches a **terminal configuration**.

  complete dependency tree

Configurations

A configuration of an arc-standard parser has three parts:

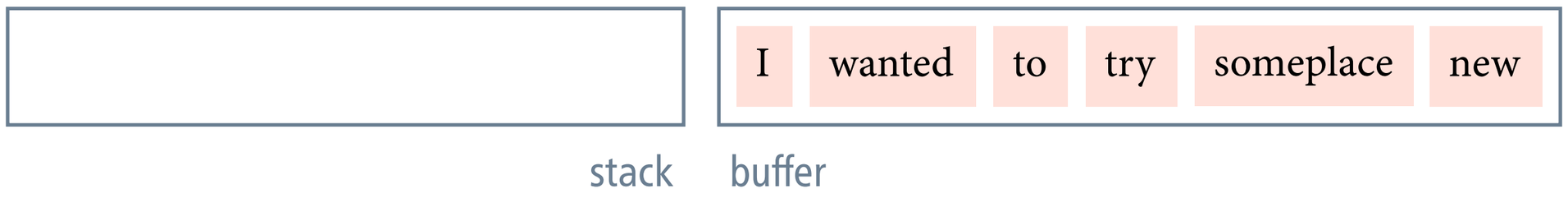
- A **buffer**, which contains those words in the sentence that still need to be processed. Initially, the buffer contains all words.
- A **stack**, which contains those words in the sentence that are currently being processed. Initially, the stack is empty.
- A **partial dependency tree**. Initially, this tree contains all the words of the sentence, but no dependency arcs.

Transitions

- The **shift transition** (**SH**) removes the frontmost word from the buffer and pushes it to the top of the stack.
- The **left-arc transition** (**LA**) creates a dependency from the topmost word on the stack to the second-topmost word, and pops the second-topmost word.
- The **right-arc transition** (**RA**) creates a dependency from the second-topmost word on the stack to the topmost word, and pops the topmost word.

Example run

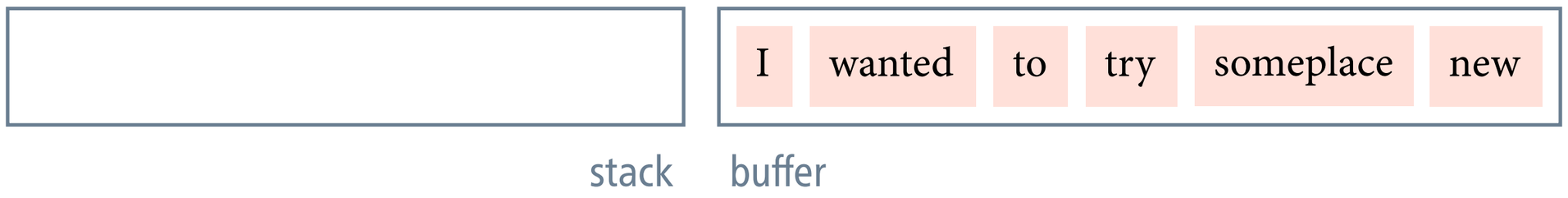
I wanted to try someplace new



(initial configuration)

Example run

I wanted to try someplace new



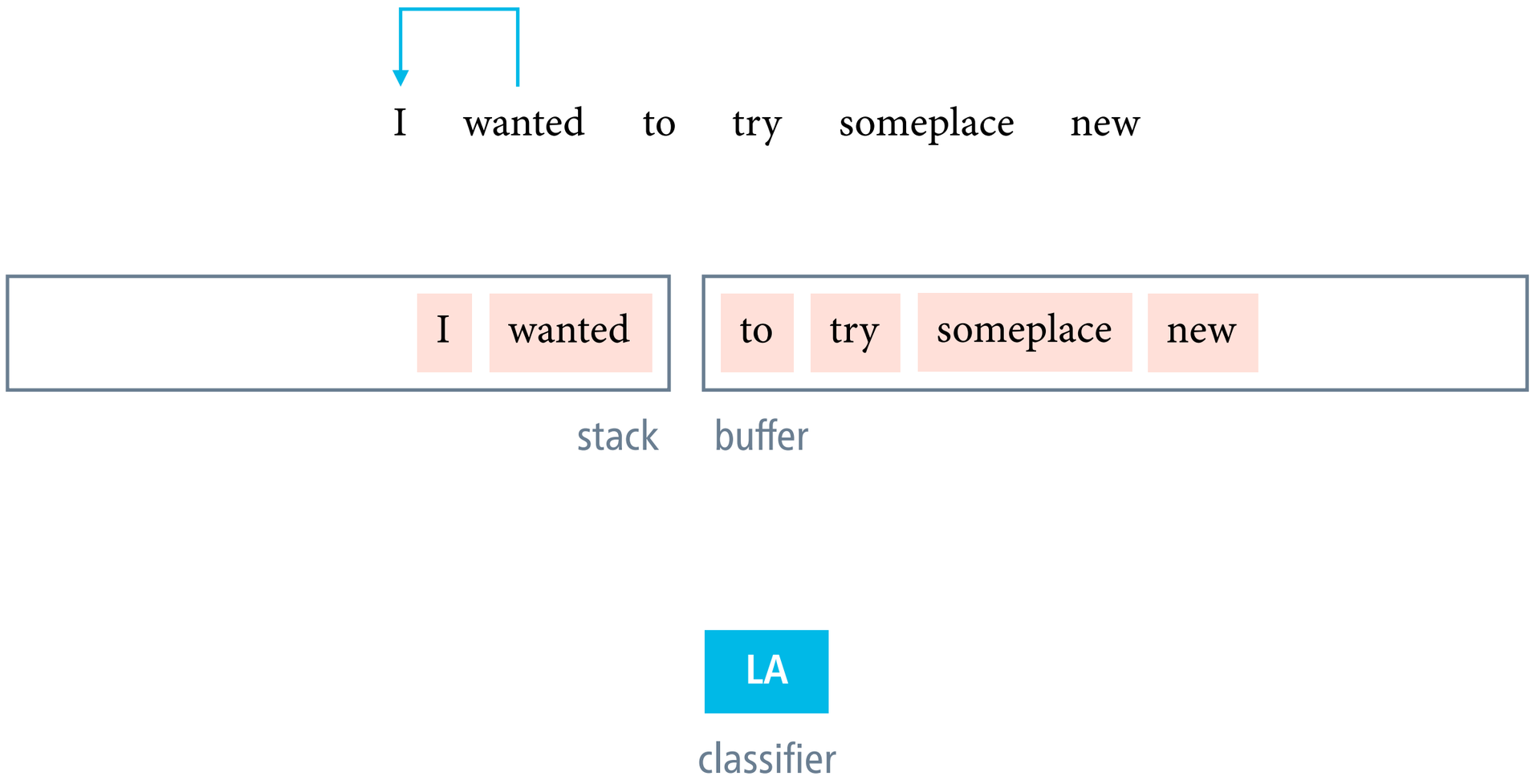
SH

classifier

Example run

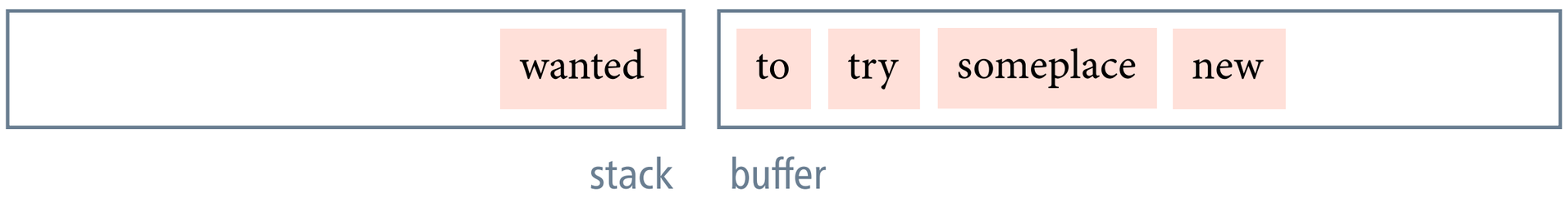
I wanted to try someplace new

Example run


I wanted to try someplace new
I
wanted
to
try
someplace
new
stack
buffer
LA
classifier

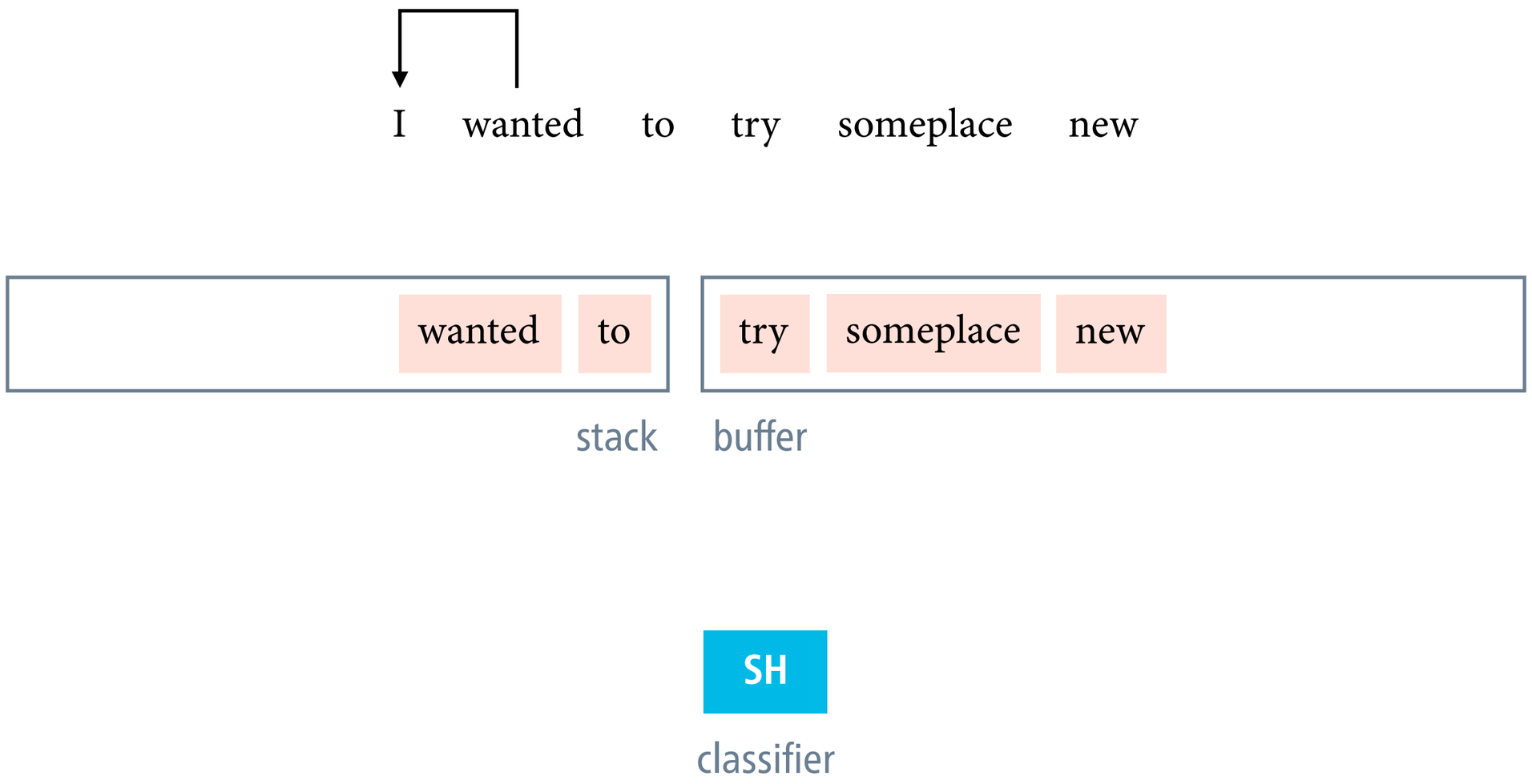
Example run


I wanted to try someplace new

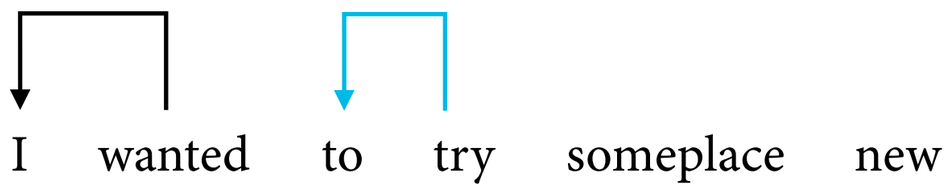

wanted
to try someplace new
stack
buffer

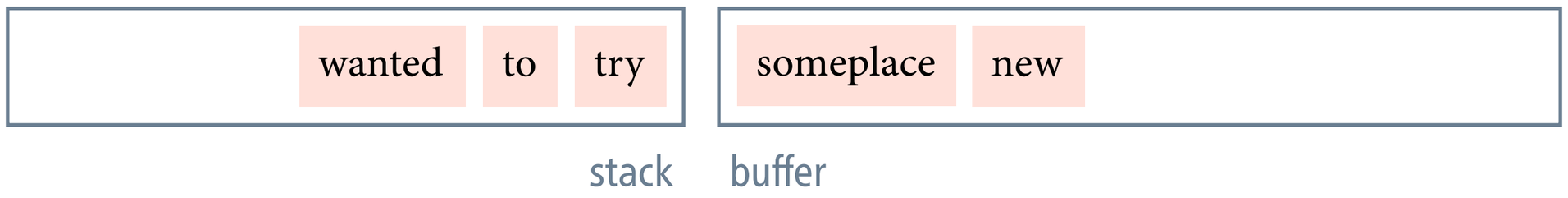

SH
classifier

Example run


I wanted to try someplace new
wanted
to
try
someplace
new
stack
buffer
SH
classifier

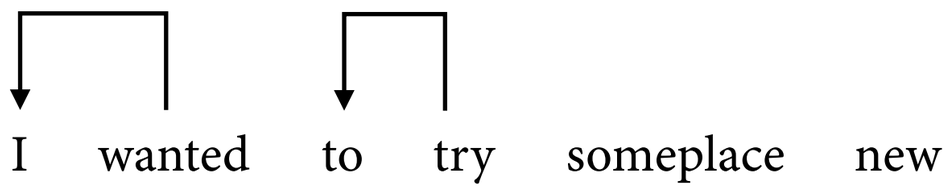
Example run

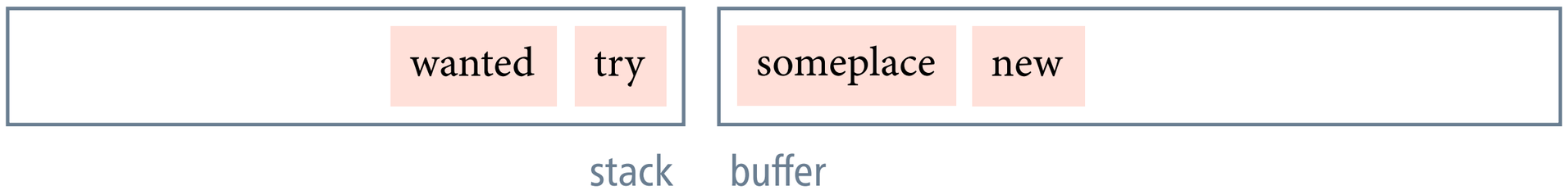

I wanted to try someplace new


wanted
to
try
someplace
new
stack
buffer


LA
classifier

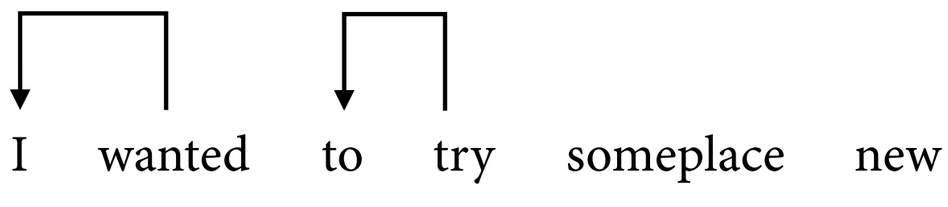
Example run

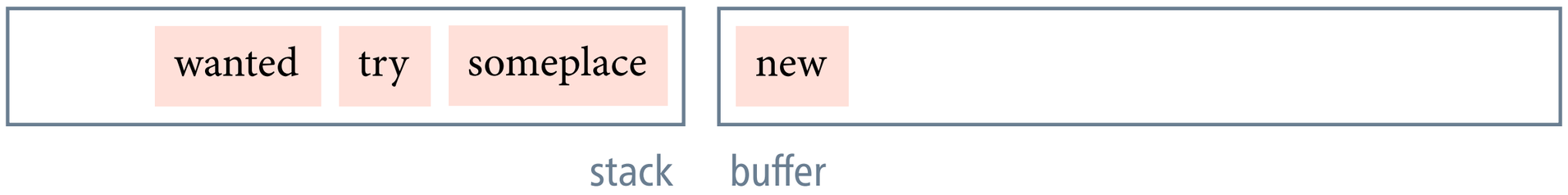

I wanted to try someplace new


wanted
try
someplace
new
stack
buffer


SH
classifier

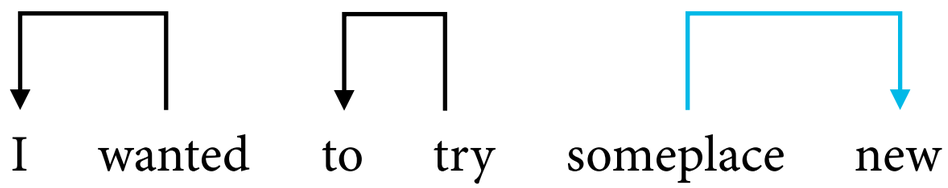
Example run

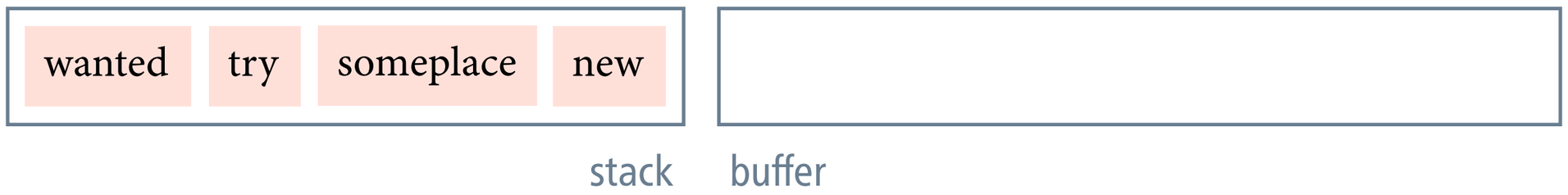

I wanted to try someplace new


wanted
try
someplace
new
stack
buffer


SH
classifier

Example run

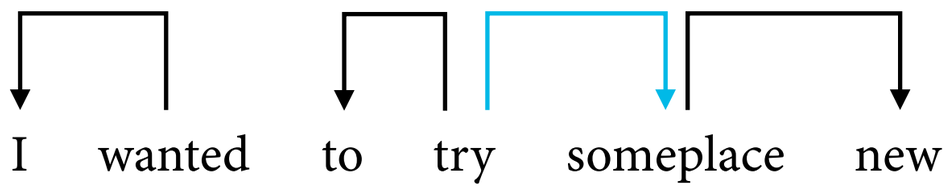


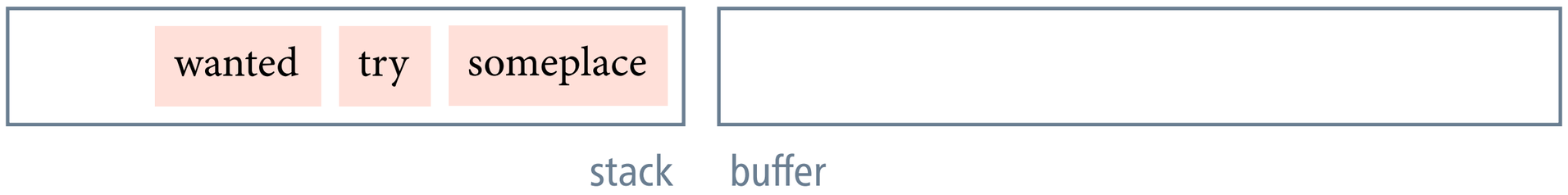


RA

classifier

Example run

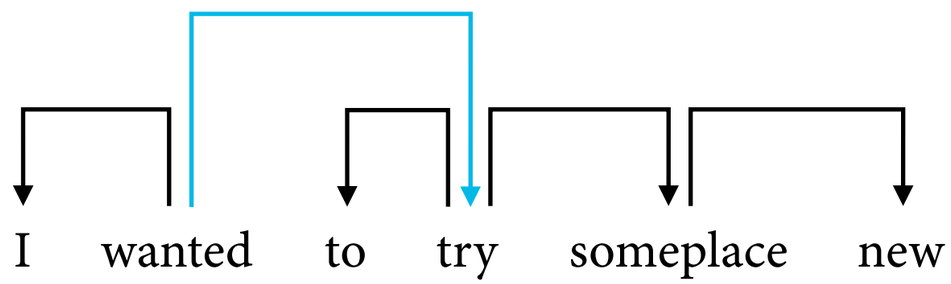


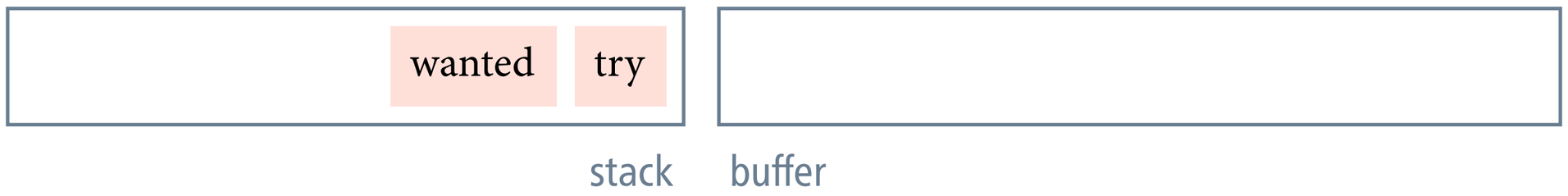


RA

classifier

Example run

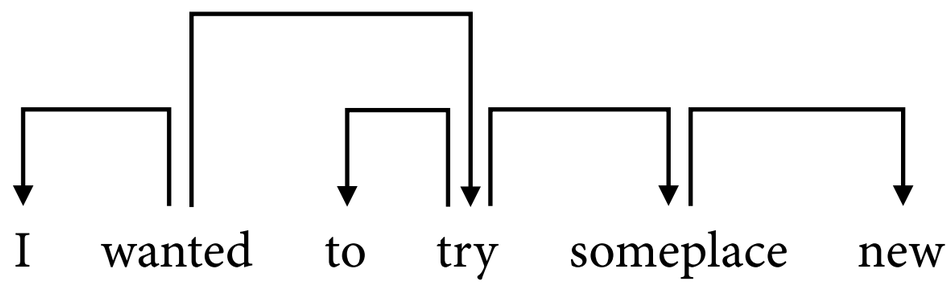


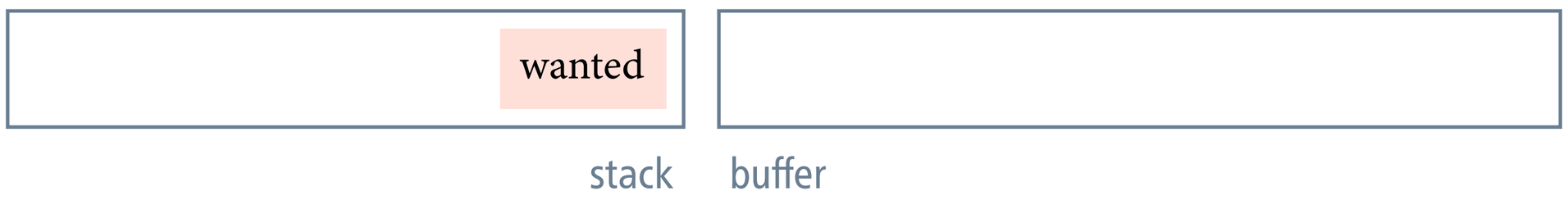


RA

classifier

Example run





(terminal configuration)

Valid transitions

Valid transitions

- SH is valid if the buffer contains at least one word.
- LA and RA are valid if the stack contains at least two words.

Valid transition sequences

are transition sequences in which all transitions are valid

Soundness and completeness

- **Soundness**

  Every valid transition sequence that starts in the initial configuration and ends in some terminal configuration builds some projective dependency tree.

- **Completeness**

  Every projective dependency tree can be built by some valid transition sequence that starts in the initial configuration and ends in some terminal configuration.

Non-uniqueness and runtime

- **Non-uniqueness**

  One and the same projective dependency tree can in general be built by several valid transition sequences.

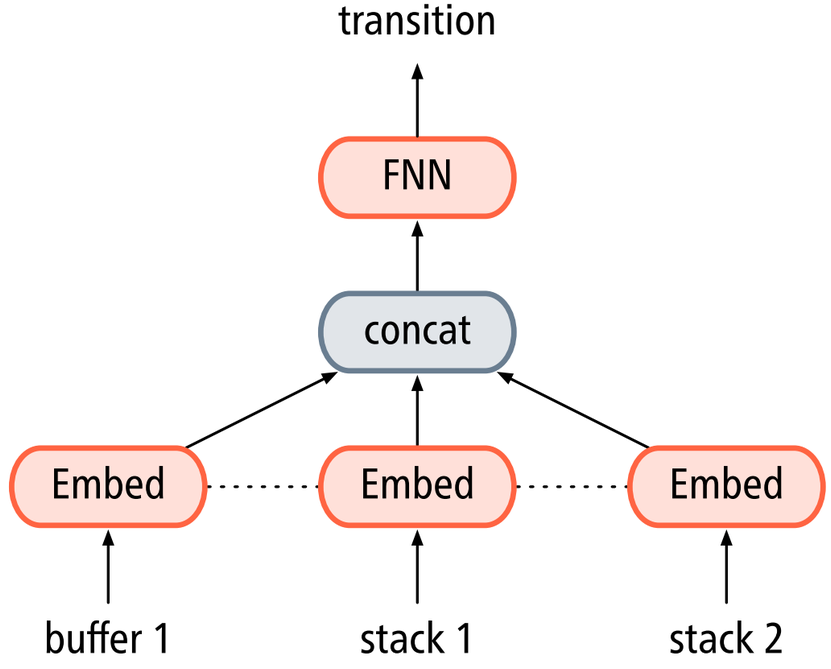
- **Runtime**

  The number of transitions that the arc-standard algorithm takes to build a tree for a sentence with n words is $2n - 1$.

Features used with the arc-standard algorithm

Features for the classifier can be defined over

- the words in the buffer
- the words on the stack
- the partial dependency tree



Chen and Manning (2014)

Static training oracle

- Choose LA if this would create an arc from the gold-standard tree, and if all arcs from the second-topmost word on the stack have already been assigned by the parser.

- Choose RA if this would create an arc from the gold-standard tree, and if all arcs from the topmost word on the stack have already been assigned by the parser.

- Otherwise, choose SH.

  must always be valid, unless the tree is non-projective