ОСНОВЫ ОБРАБОТКИ ТЕКСТОВ

Лекция #8:
Синтаксический анализ (часть 2)

Лектор: м.н.с. ИСП РАН Майоров Владимир Дмитриевич

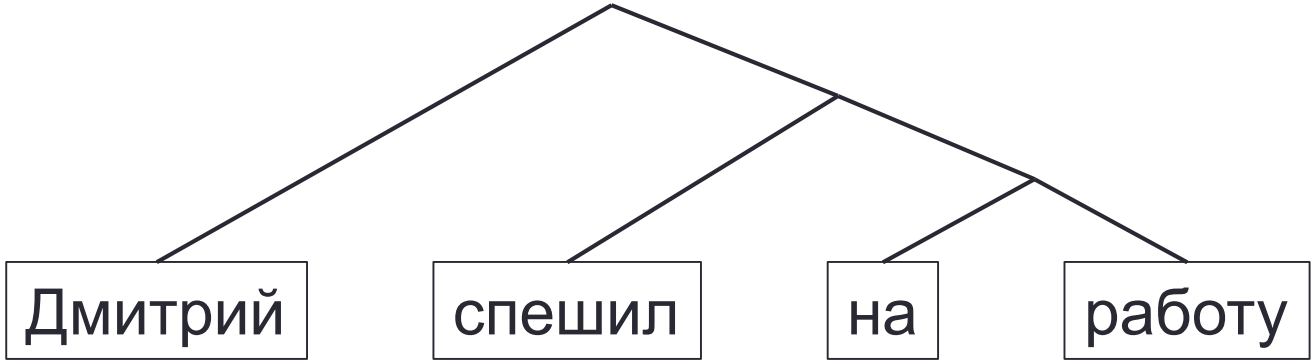
Синтаксис

- Предложение – это единица языка, которая представляет собой грамматически организованное соединение слов, обладающее смысловой законченностью.

- Грамматика – раздел лингвистики, который изучает закономерности построения правильных осмысленных речевых отрезков (словоформ, словосочетаний, предложений, текстов).
- Синтаксис — раздел лингвистики, изучающий и моделирующий правила, по которым образуются единицы, более крупные, чем слово, а именно словосочетания и предложения.

Синтаксическая структура

- Представление предложения в виде вложенных составляющих.
- Составляющая (фраза, синтаксическая группа) – структурная единица предложения, составленная из более тесно связанных друг с другом составляющих меньшего размера.



Синтаксические правила

- Существительное управляет прилагательным

красная книга → красная книга

красный книга → не словосочетание

- Предлог управляет существительным

в книга → не словосочетание

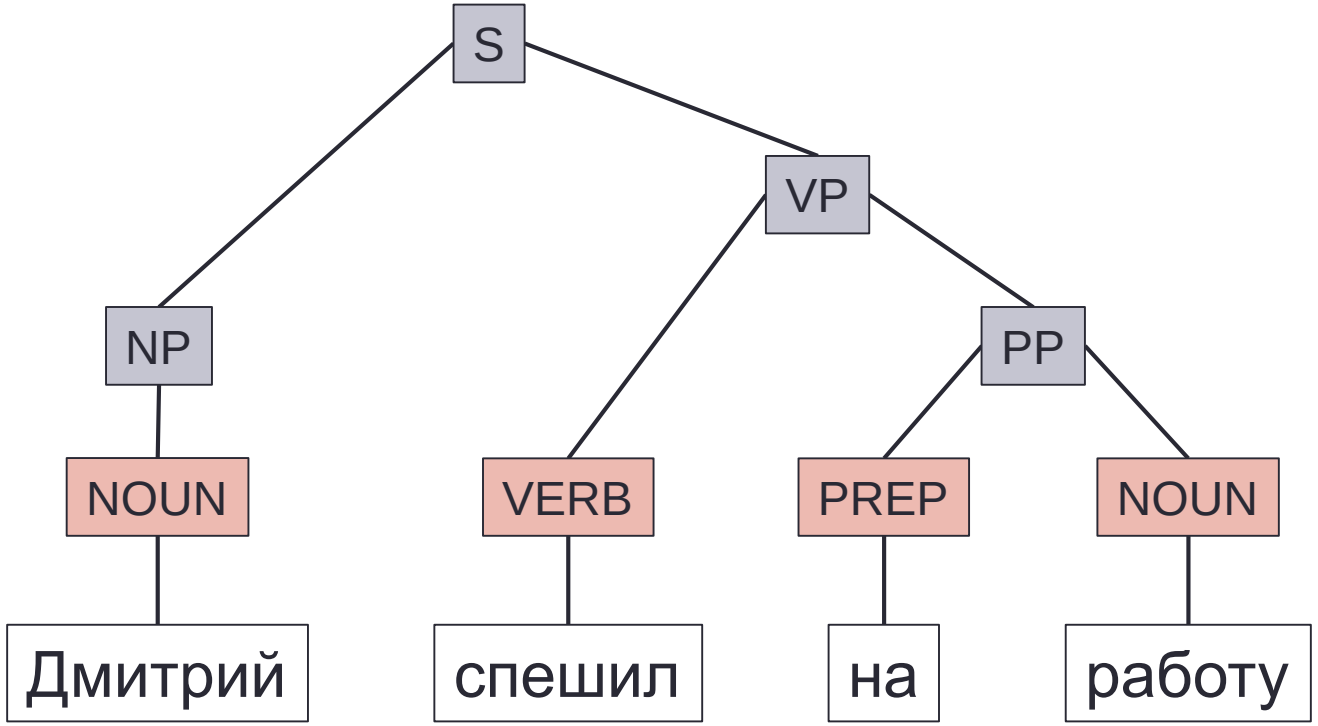
в книге → в книге

Синтаксические правила

- В зависимости от главного слова в словосочетании выделяют
  - Именные группы (главное – существительное)
  - Группа прилагательного
  - Наречная группа
  - Предложная группа
  - Глагольная группа
  - Предложение

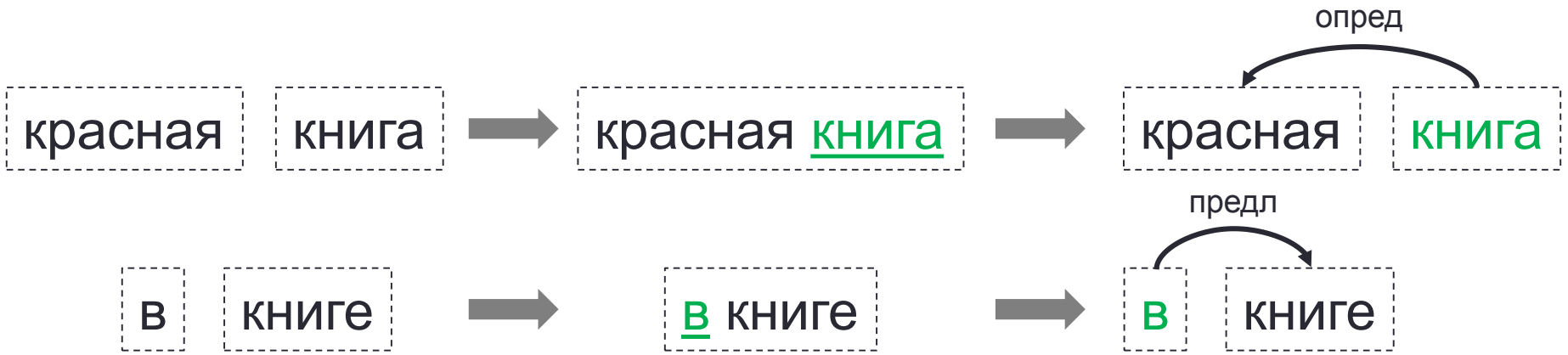
Синтаксическая структура

- Каждой составляющей назначается тип в зависимости от главного слова



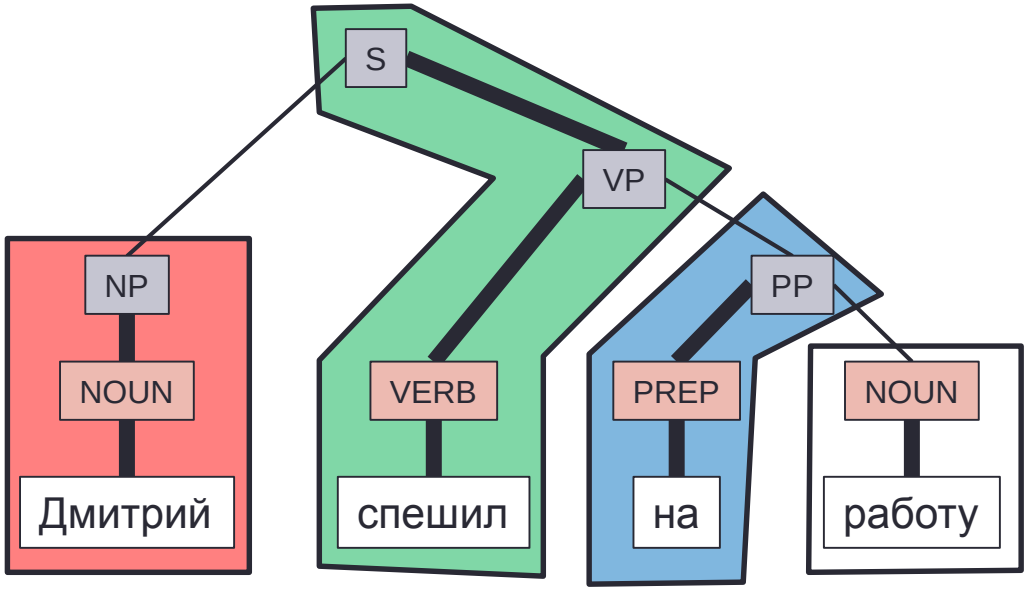
Грамматика зависимостей

- Формальная модель описания структуры словосочетаний (предложений)
- Описывается в виде иерархии слов, между которыми установлено бинарное отношение зависимости
- Различают типы зависимостей (соответствуют типам синтаксических правил)



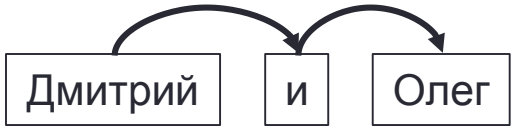
Грамматика зависимостей

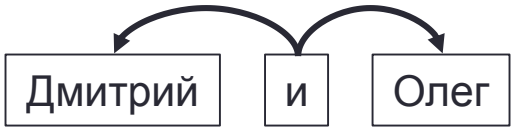
- Дерево составляющих может быть преобразовано в дерево зависимостей

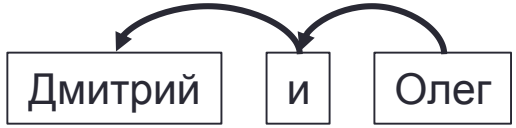


Грамматика зависимостей

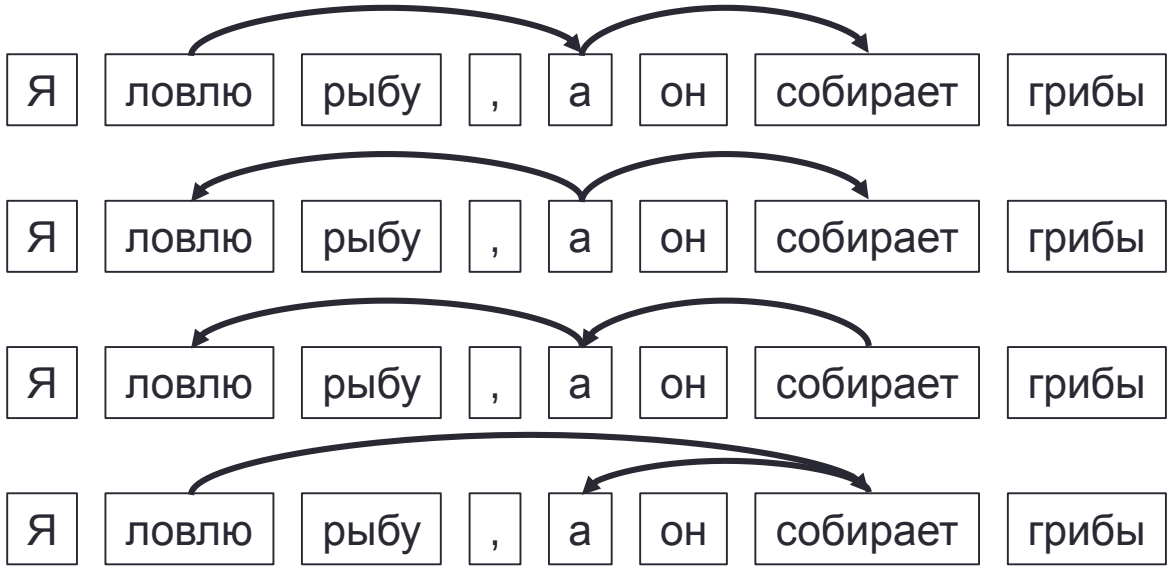
- Не все синтаксические отношения подчинительные
  - Сочинительная группа





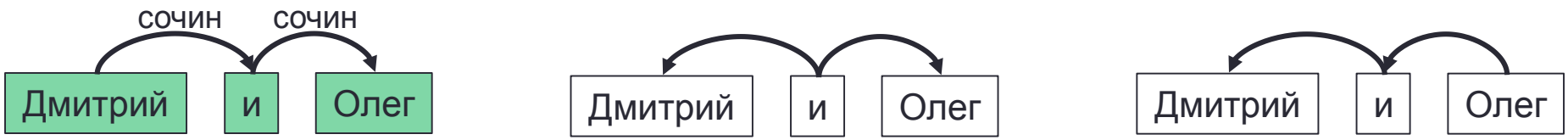


  - Сложносочиненные предложения

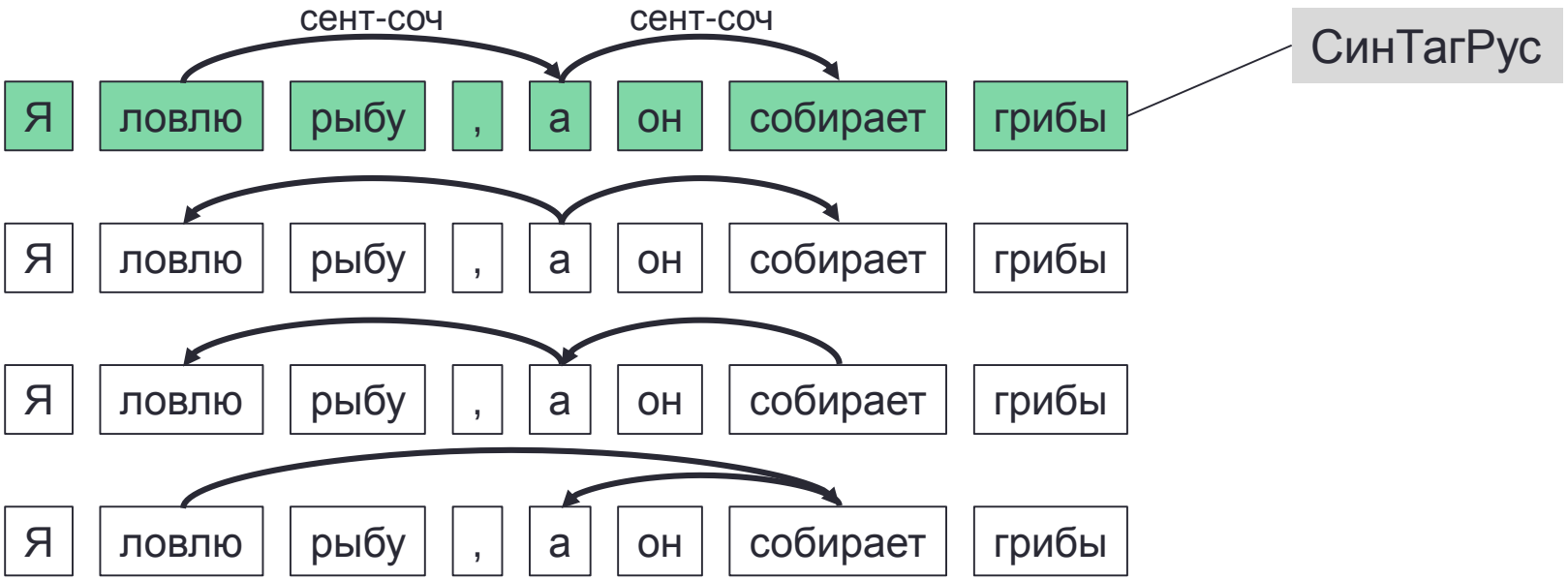


Грамматика зависимостей

- Не все синтаксические отношения подчинительные
  - Сочинительная группа

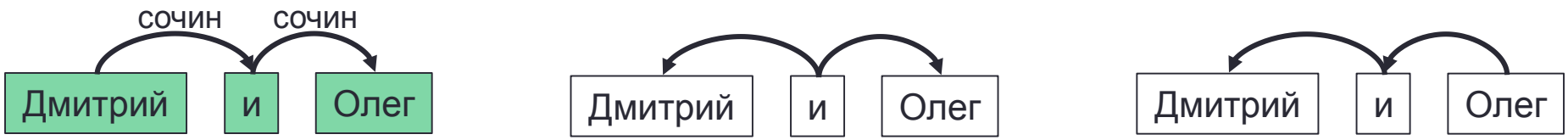


  - Сложносочиненные предложения

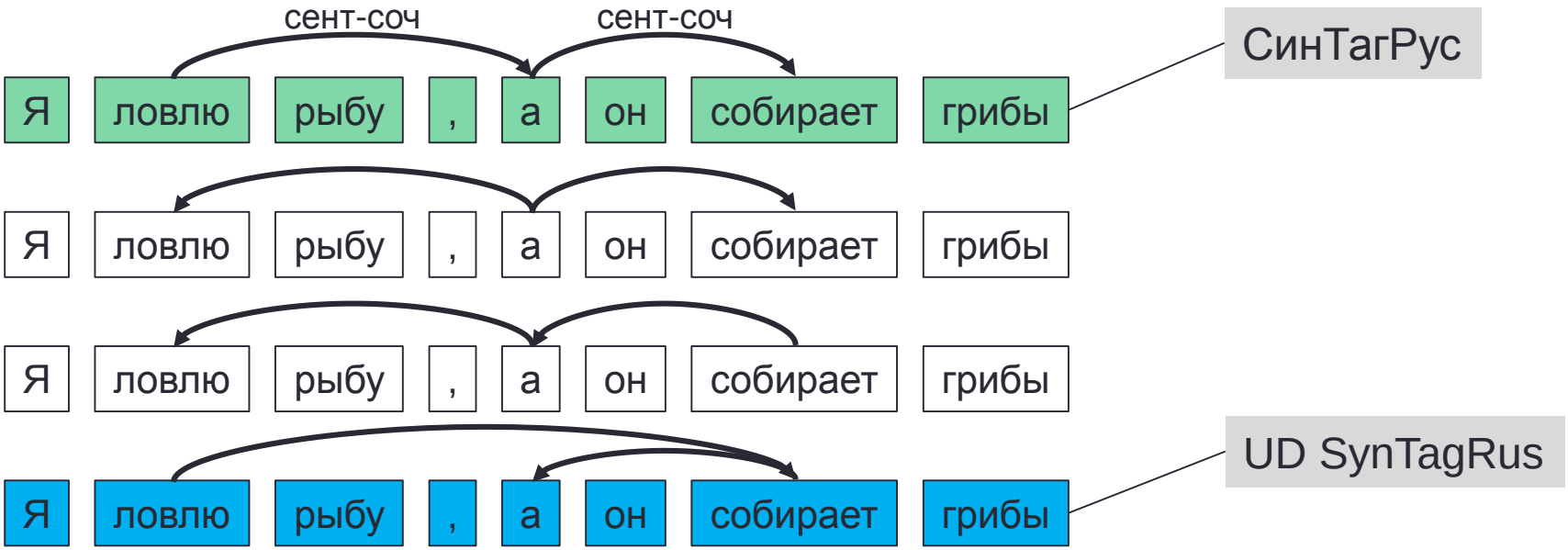


Грамматика зависимостей

- Не все синтаксические отношения подчинительные
  - Сочинительная группа



  - Сложносочиненные предложения



Universal Dependencies

- 83 языка
  - Универсальный набор грамматических категорий
  - Универсальный набор синтаксических отношений
- 146 наборов данных (CoNLL-U формат)
  - Часть размечена вручную
  - Часть сконвертирована из других форматов

https://universaldependencies.org

CoNLL-U формат

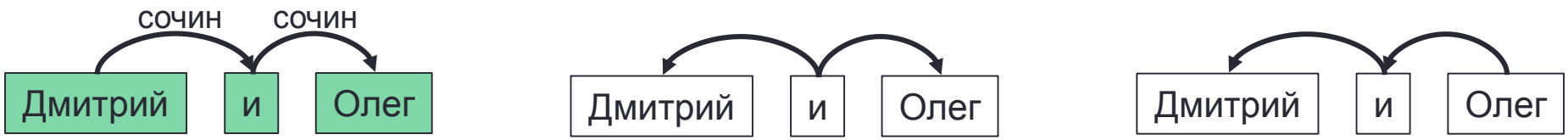
- Tab-separated формат с 10 колонками

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC

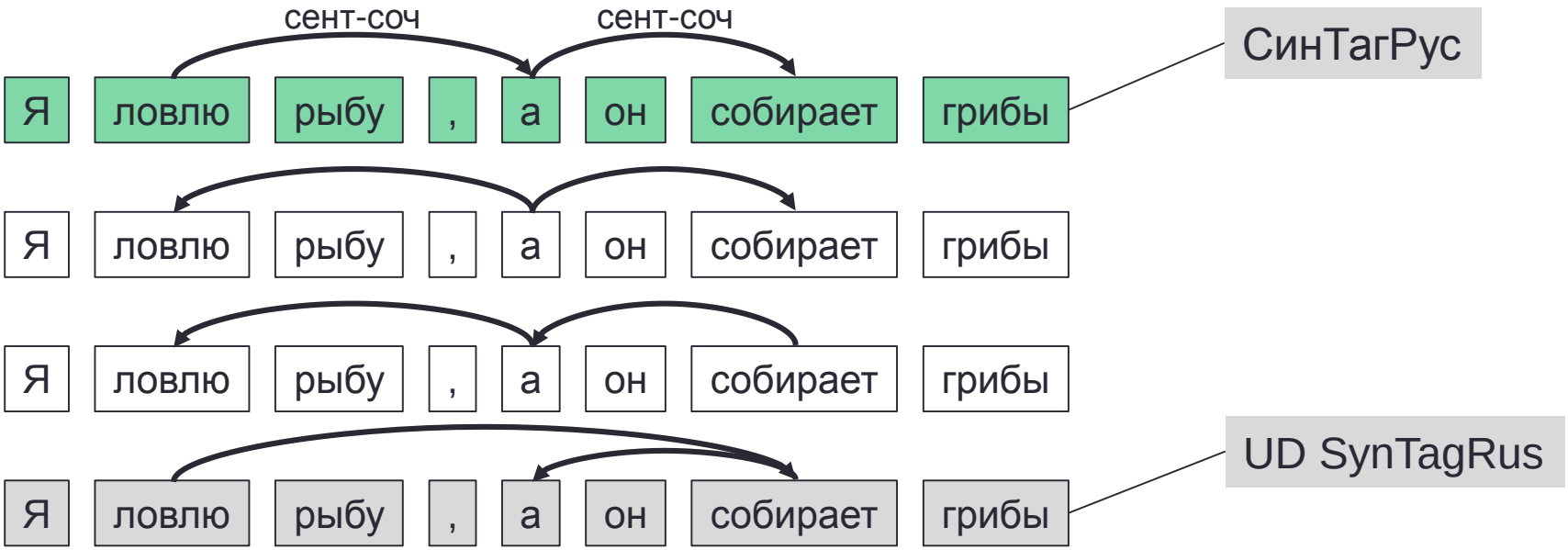
- ID – номер слова в предложении (с 1);
- FORM - слово или знак пунктуации;
- LEMMA - начальная форма слова;
- UPOS - универсальная часть речи (из списка UD);
- XPOS - уточнение части речи для конкретного языка;
- FEATS - список морфологических признаков слова;
- HEAD – ID главного слова для данного слова; 0 для корня предложения;
- DEPREL - метка зависимости от главного слова;
- DEPS - информация для усиленного графа зависимостей;
- MISC - любые дополнительные сведения

Грамматика зависимостей

- Не все синтаксические отношения подчинительные
  - Сочинительная группа



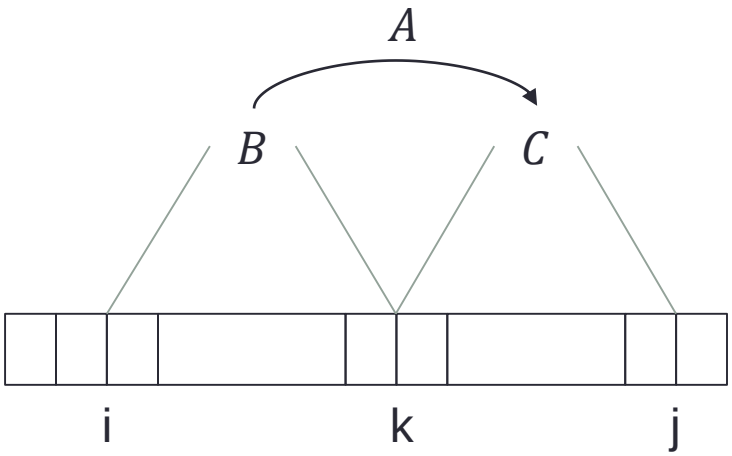
  - Сложносочиненные предложения



Методы динамического программирования

- Некоторое обобщение алгоритма CKY для грамматик зависимостей
  - Строится дерево разбора предложения снизу-вверх
  - На каждом этапе рассматриваются подстроки исходной строки
  - Если подстрока состоит из двух деревьев, оценивается левая и правая дуга между ними:
    - $score(A) = score(B) + score(C) + score(arc)$

- Варианты
  - Алгоритм Коллинса. Сложность $O(n^5)$
  - Алгоритм Эйснера. Сложность $O(n^3)$



Transition-based parsing

- *Система переходов* состоит из
  - C – множество конфигураций
  - T – множество переходов. $t: C \rightarrow C$
  - c_s – функция инициализации
  - $C_t \subseteq C$ – множество конечных конфигураций

- *Конфигурация* состоит из:
  - Σ – стек
  - B – буфер
  - A – множество зависимостей (i, l, j)

- Последовательность переходов для последовательности x:
  - $c_0 = c_s(x)$
  - $c_i = t(c_{i-1}), i = \overline{1..m}$
  - $c_m \in C_t$

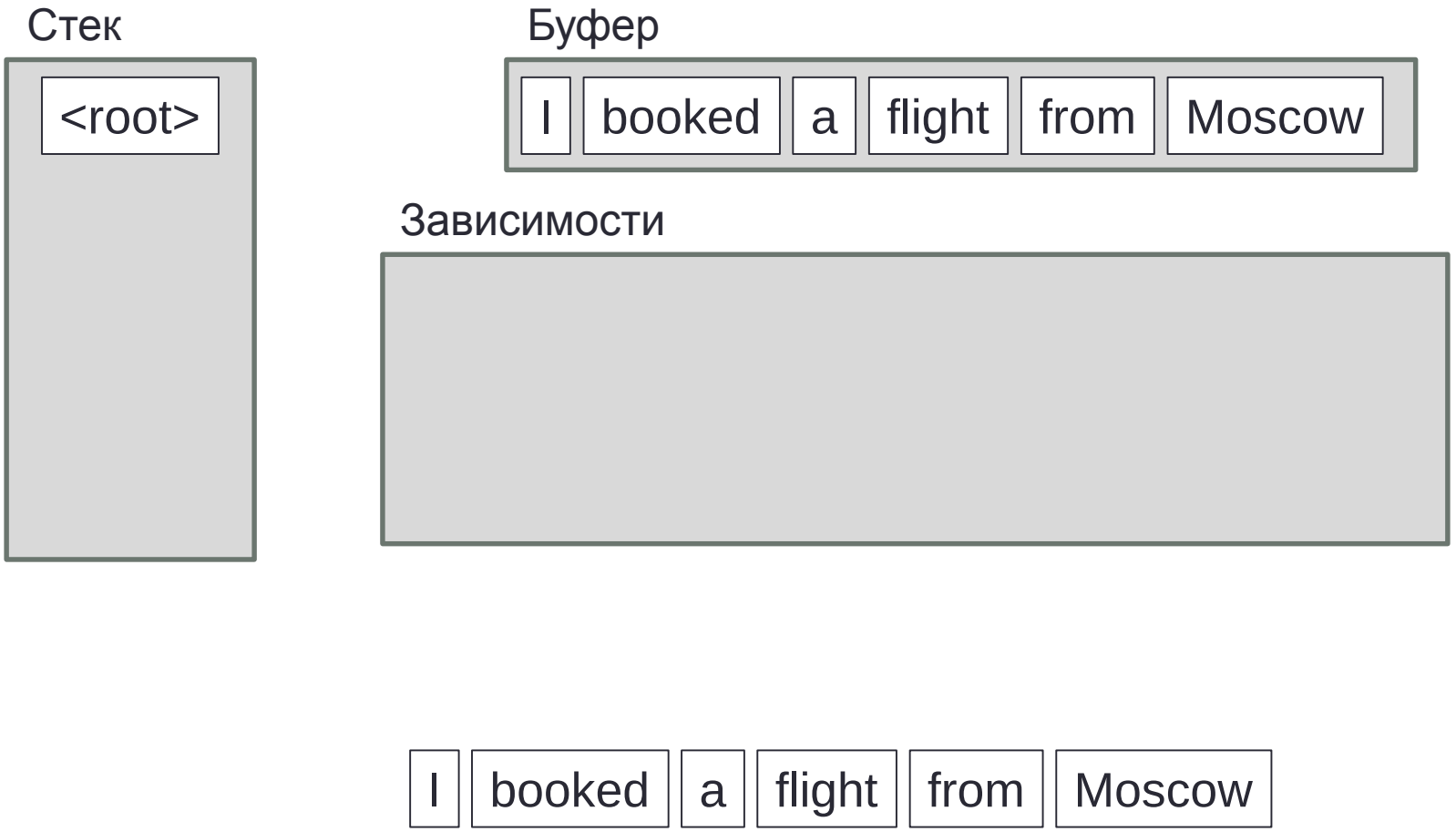
Transition-based parsing

- Алгоритм
  - Получить начальную конфигурацию $c = c_s(x)$
  - Пока $c \notin C_t$
    - Выбрать переход $t = \underset{t}{\operatorname{argmax}}\, Score(c, t)$
    - Выполнить переход $c = t(c)$
  - Извлечь множество зависимостей из $c.A$

Arc-standard

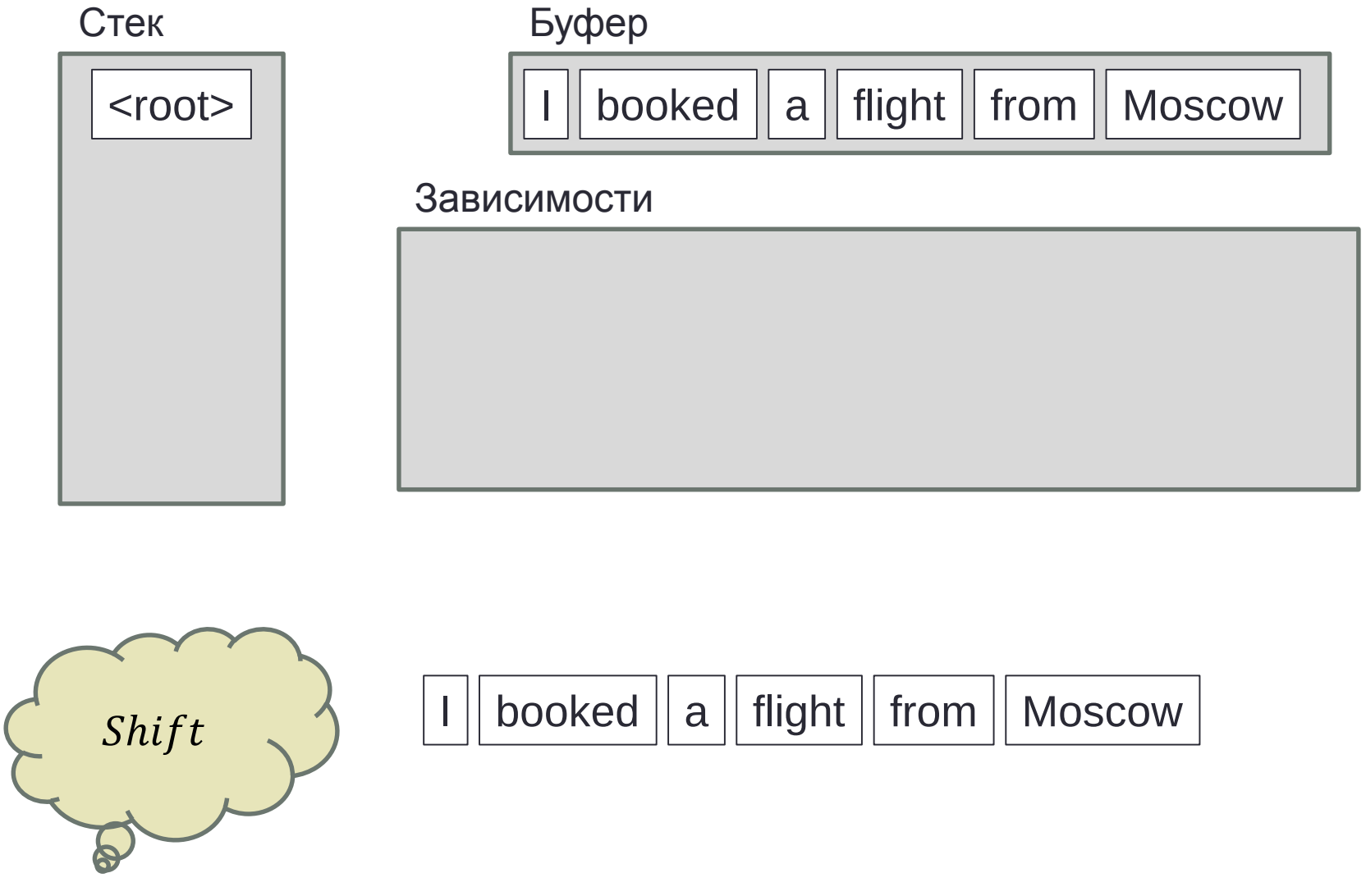
- Инициализация
  - $c_s(x = x_1 \dots x_n) = ([0], [1, \dots, n], \emptyset)$
- Конечное состояние
  - $C_t = \{c \in C | c = ([0], [\,], A)\}$
- Переходы
  - $(Shift)$ $(\sigma, [i|\beta], A) \rightarrow ([\sigma|i], \beta, A)$
  - $(LeftArc_l)$ $([\sigma|i|j], B, A) \rightarrow ([\sigma|j], B, A \cup \{(j, l, i)\})$, если $i \neq 0$
  - $(RightArc_l)$ $([\sigma|i|j], B, A) \rightarrow ([\sigma|i], B, A \cup \{(i, l, j)\})$

Arc-standard

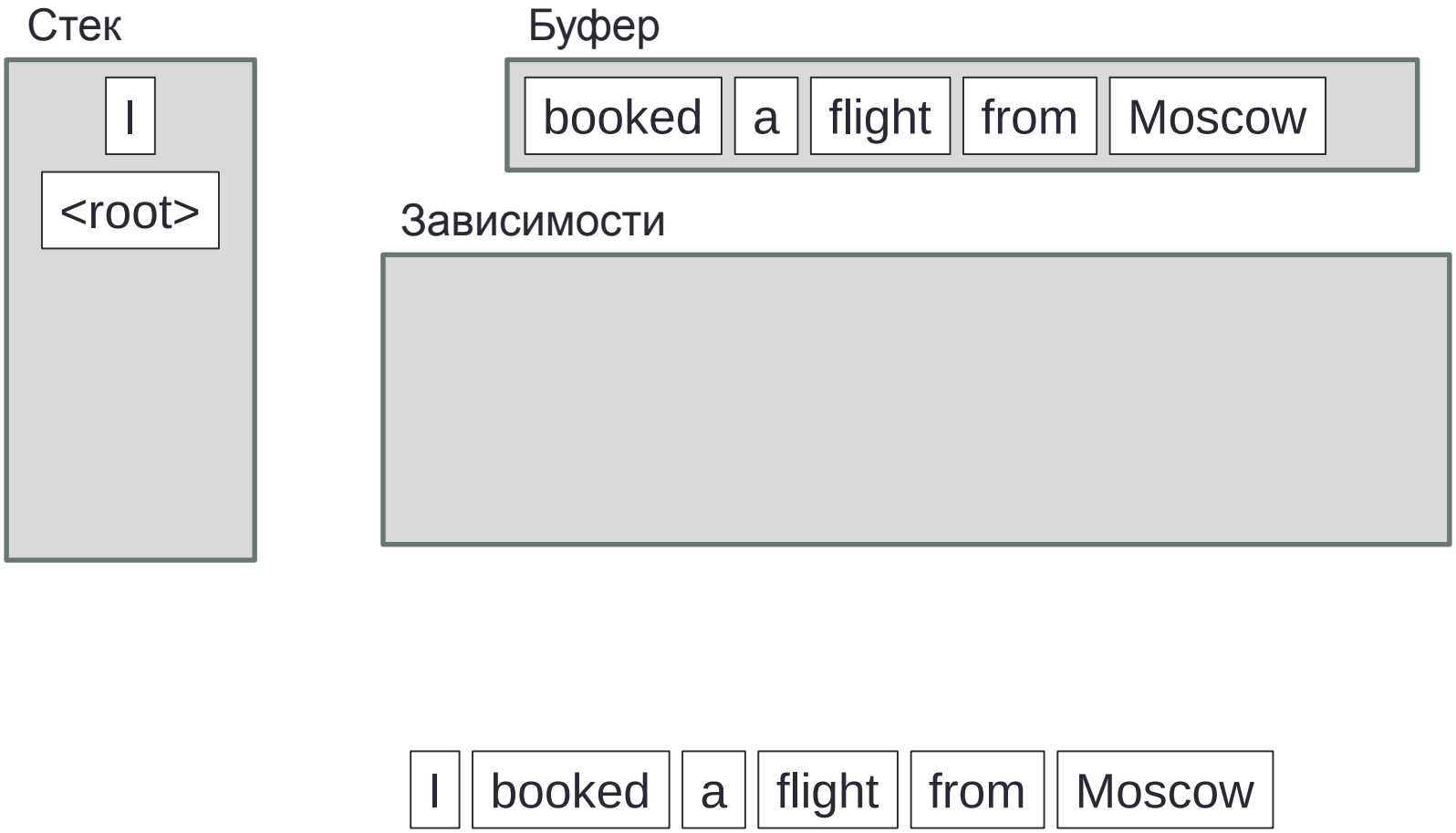


I | booked | a | flight | from | Moscow

Arc-standard

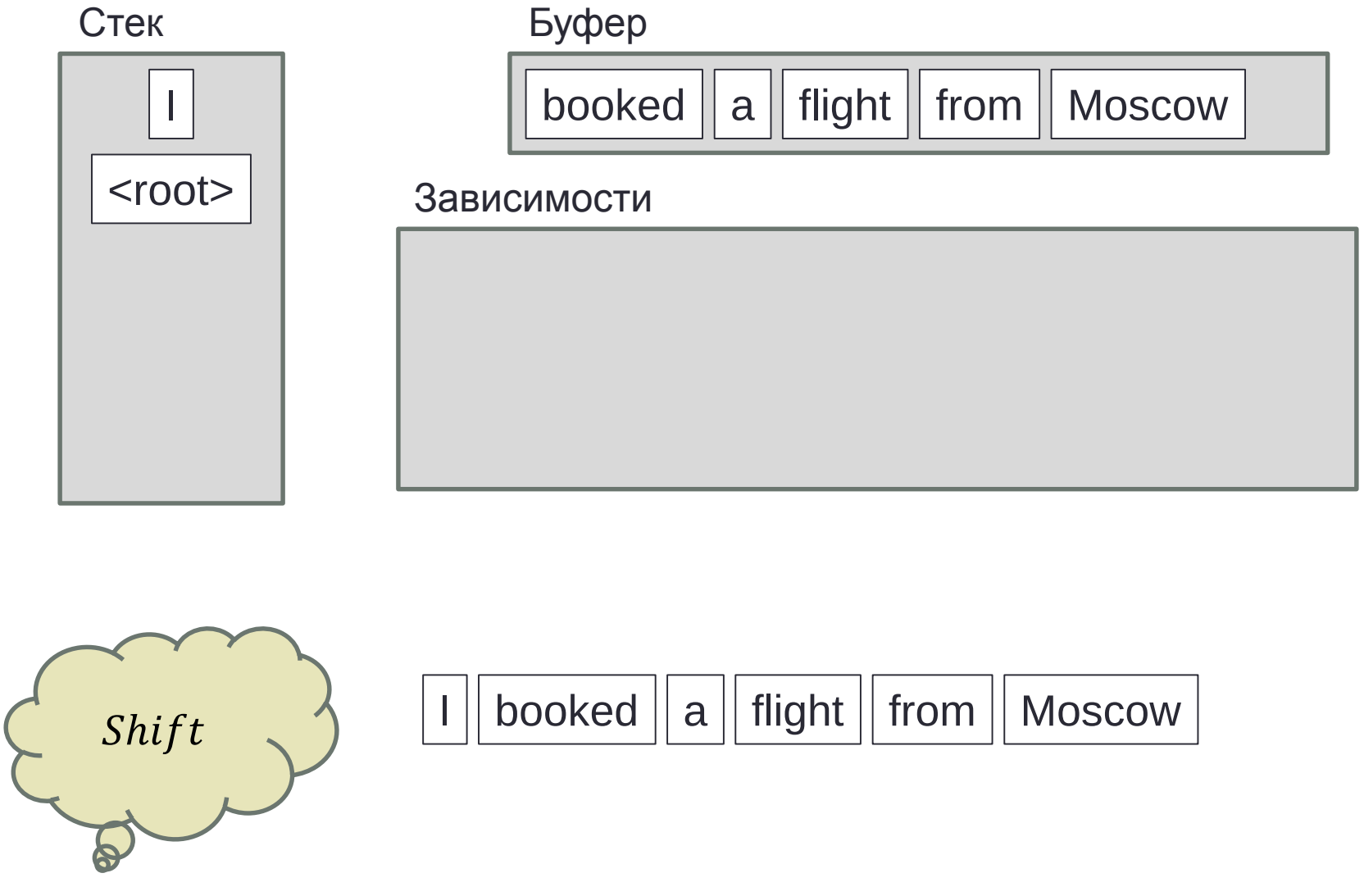


Arc-standard

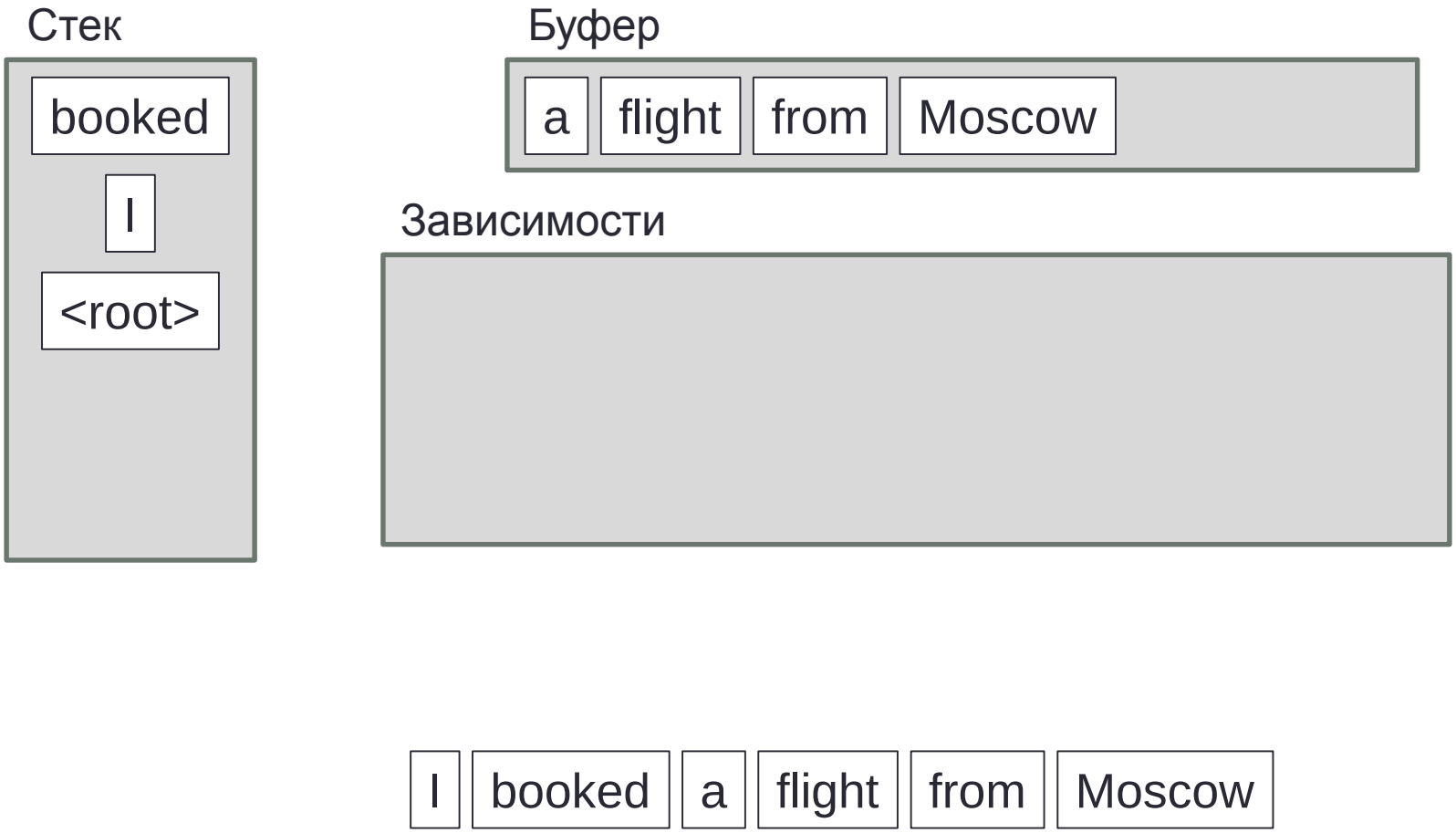


I | booked | a | flight | from | Moscow

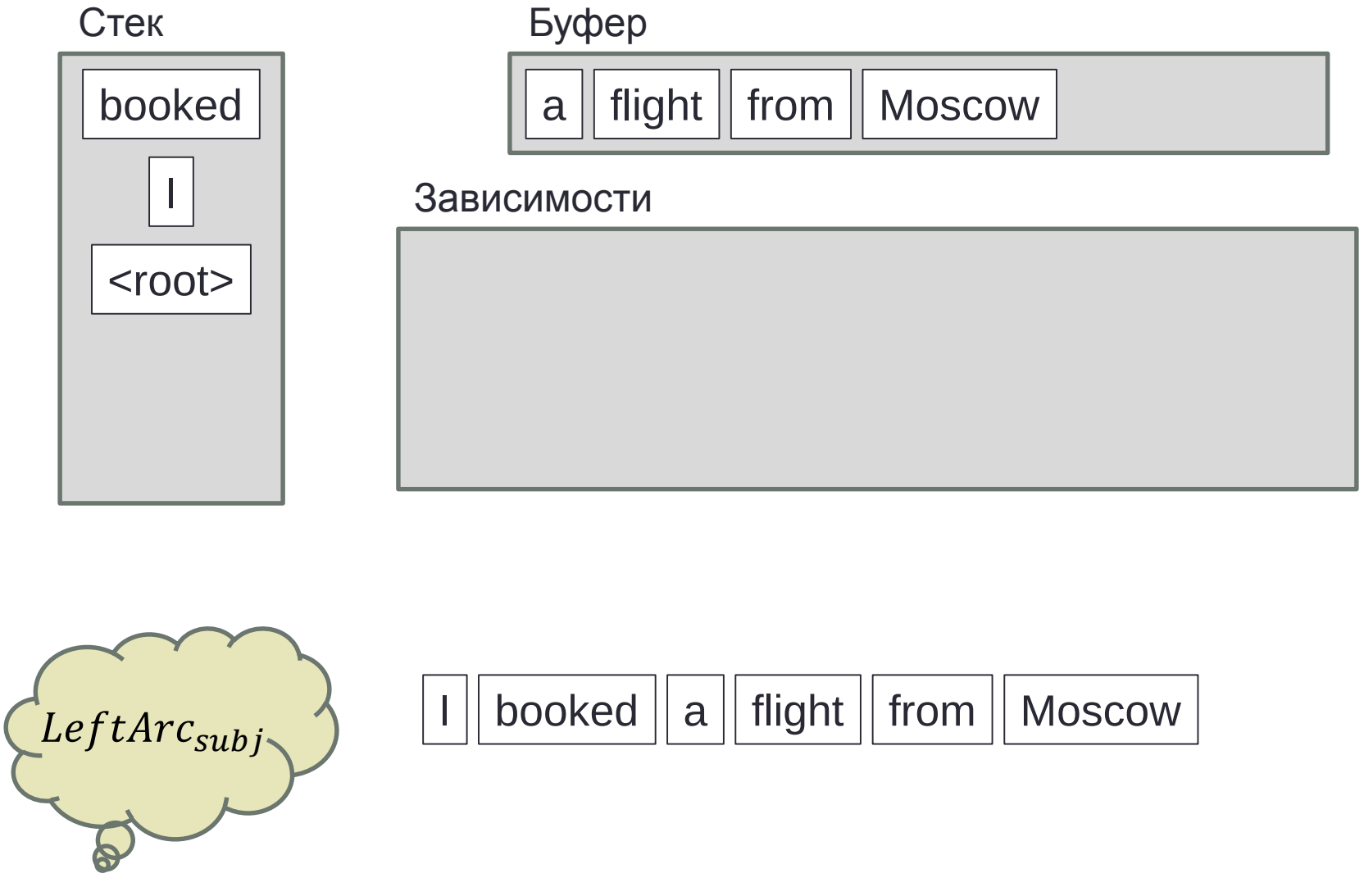
Arc-standard



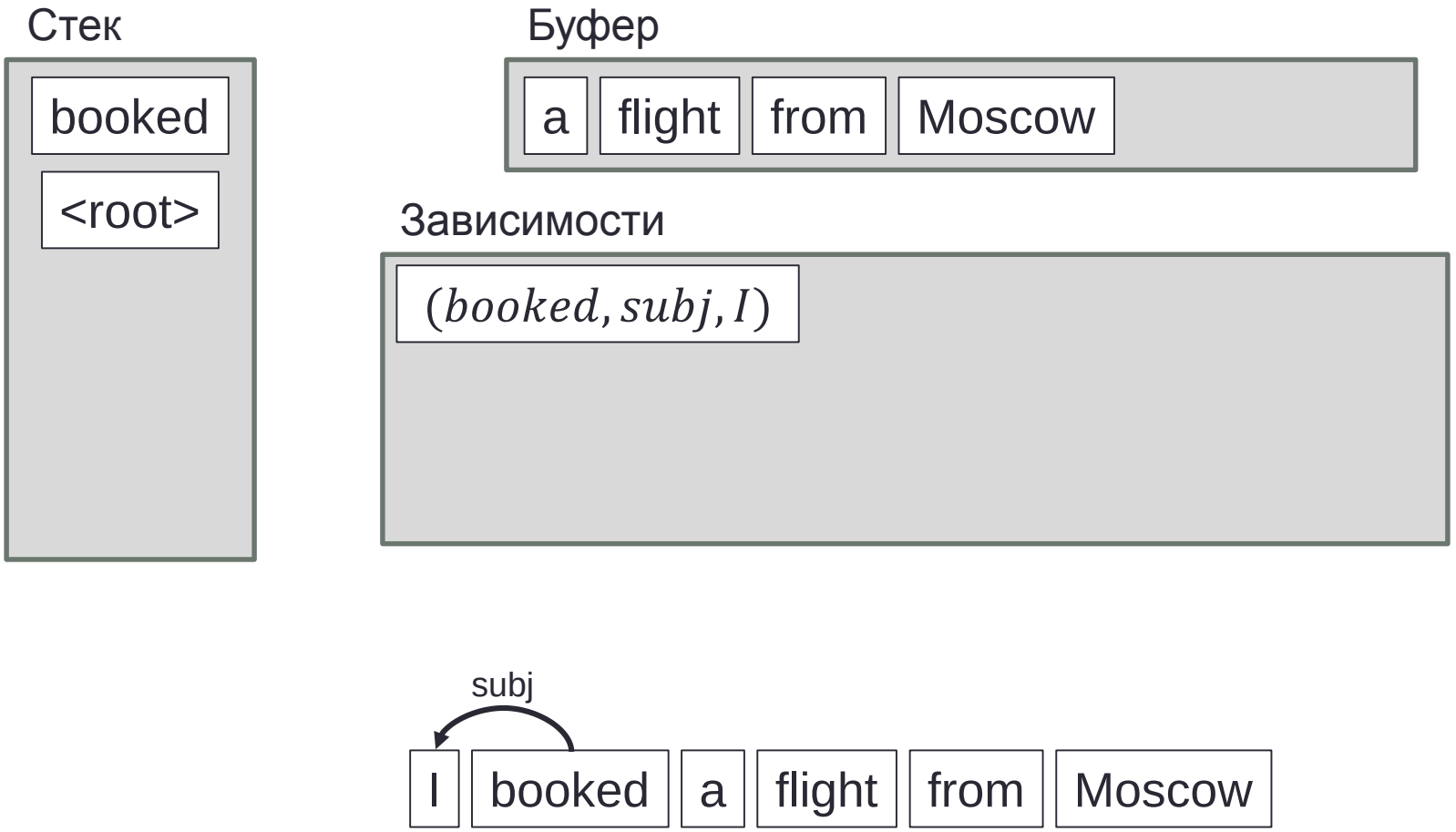
Arc-standard



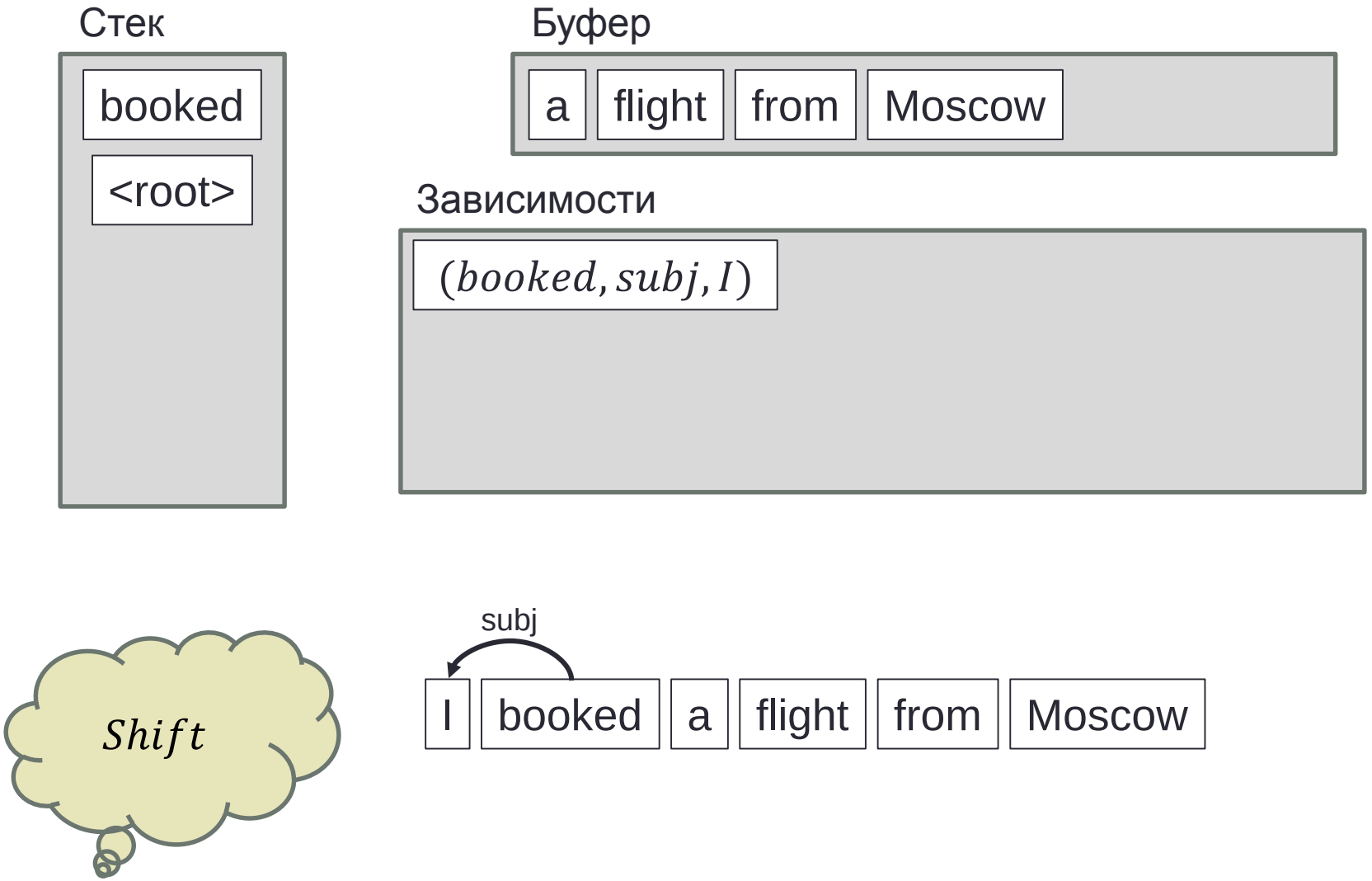
Arc-standard



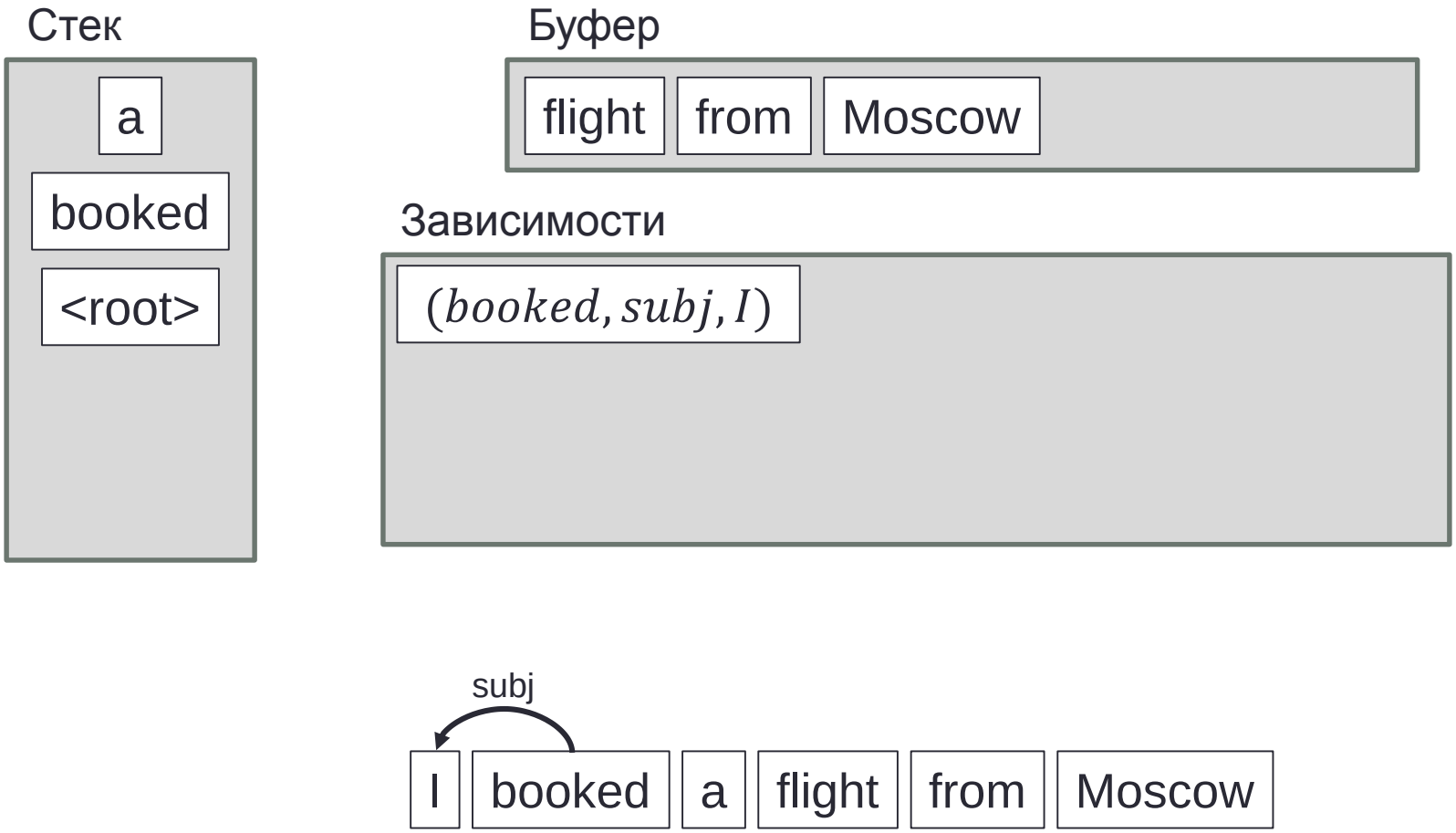
Arc-standard



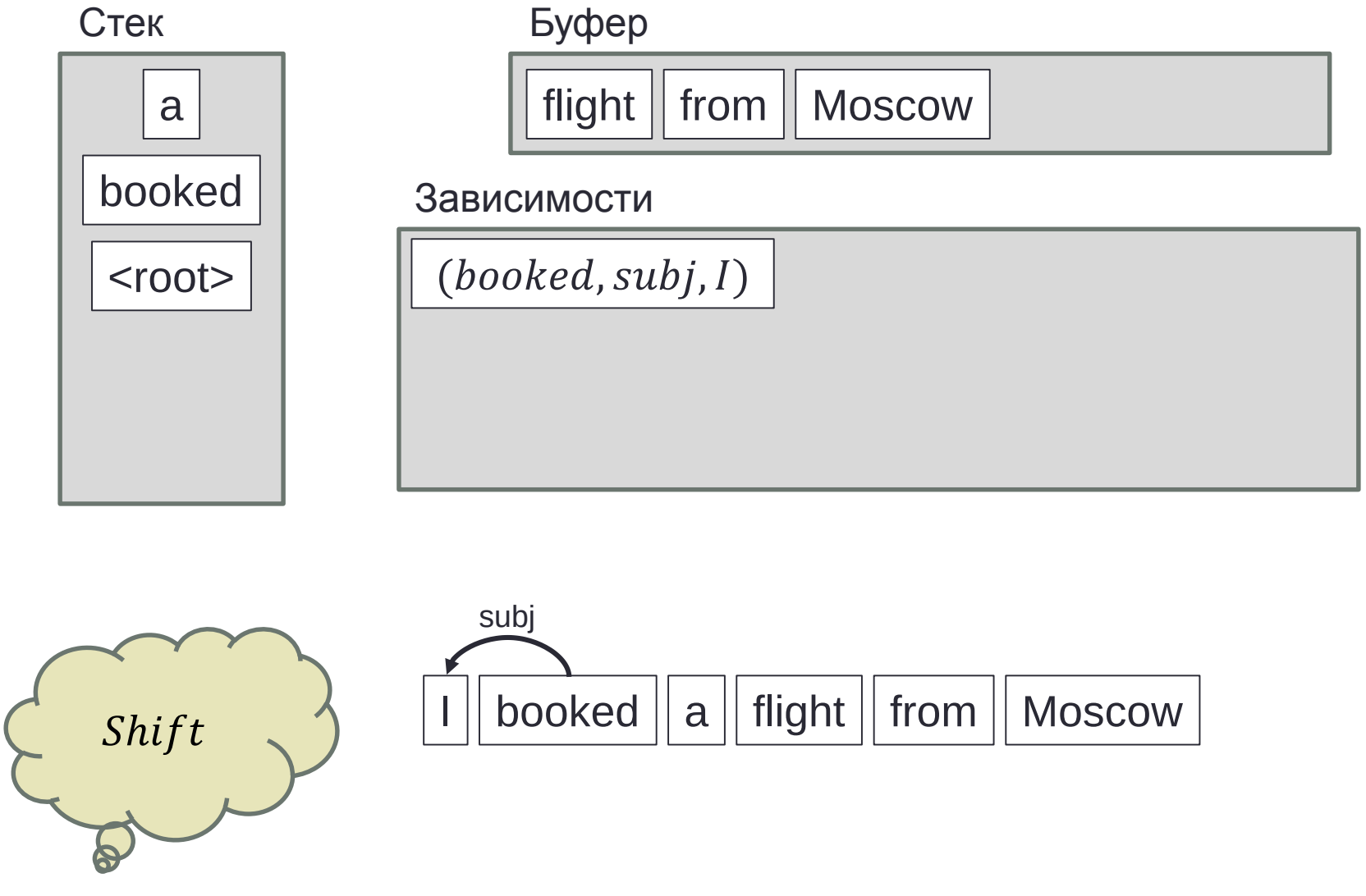
Arc-standard



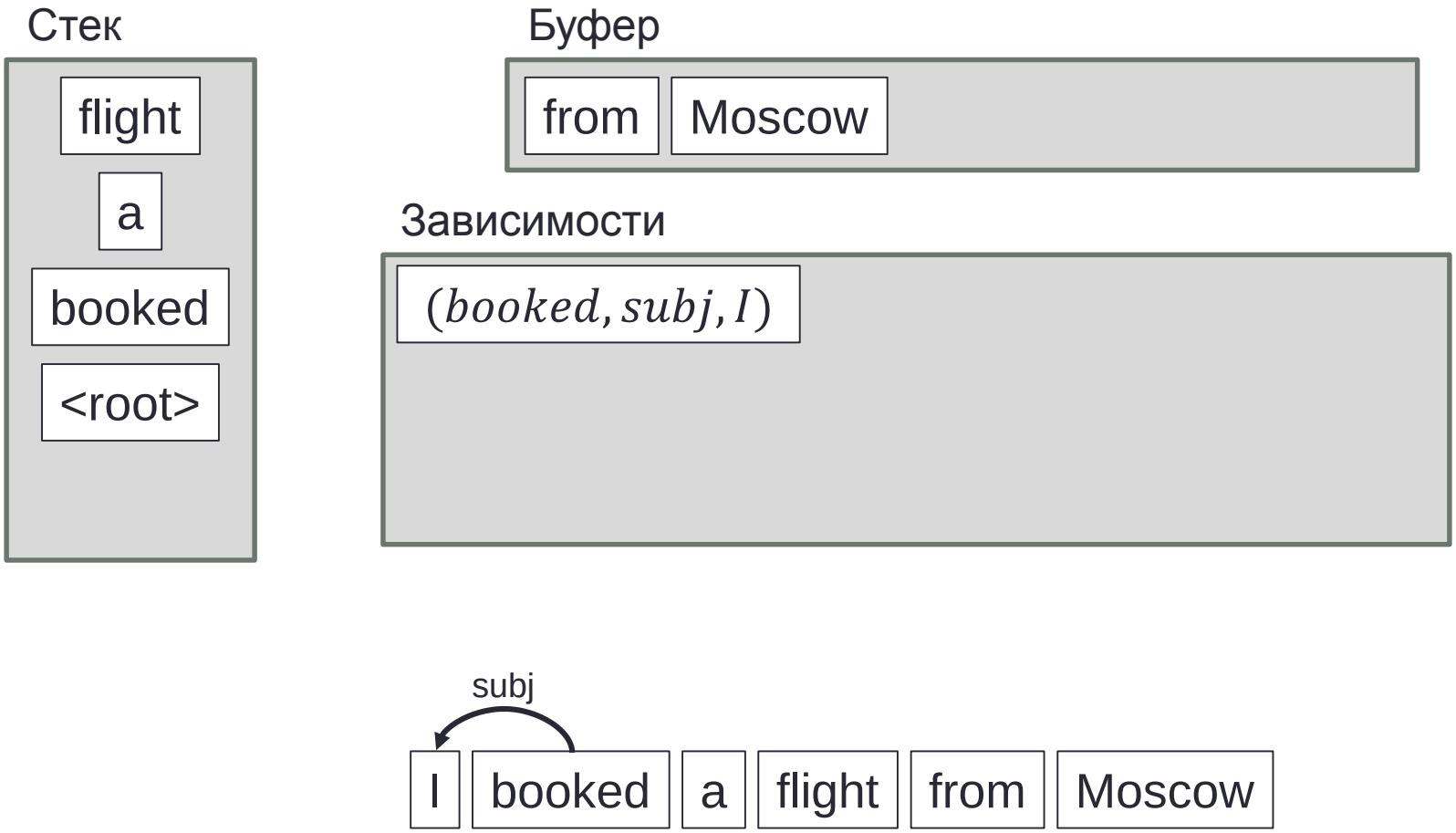
Arc-standard



Arc-standard

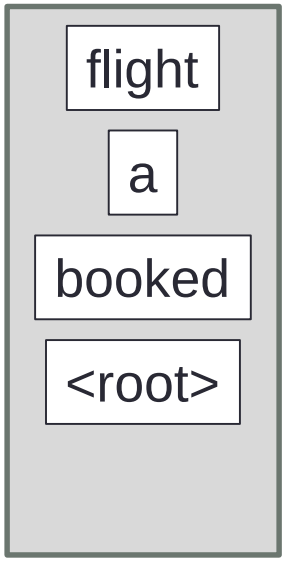


Arc-standard



Arc-standard

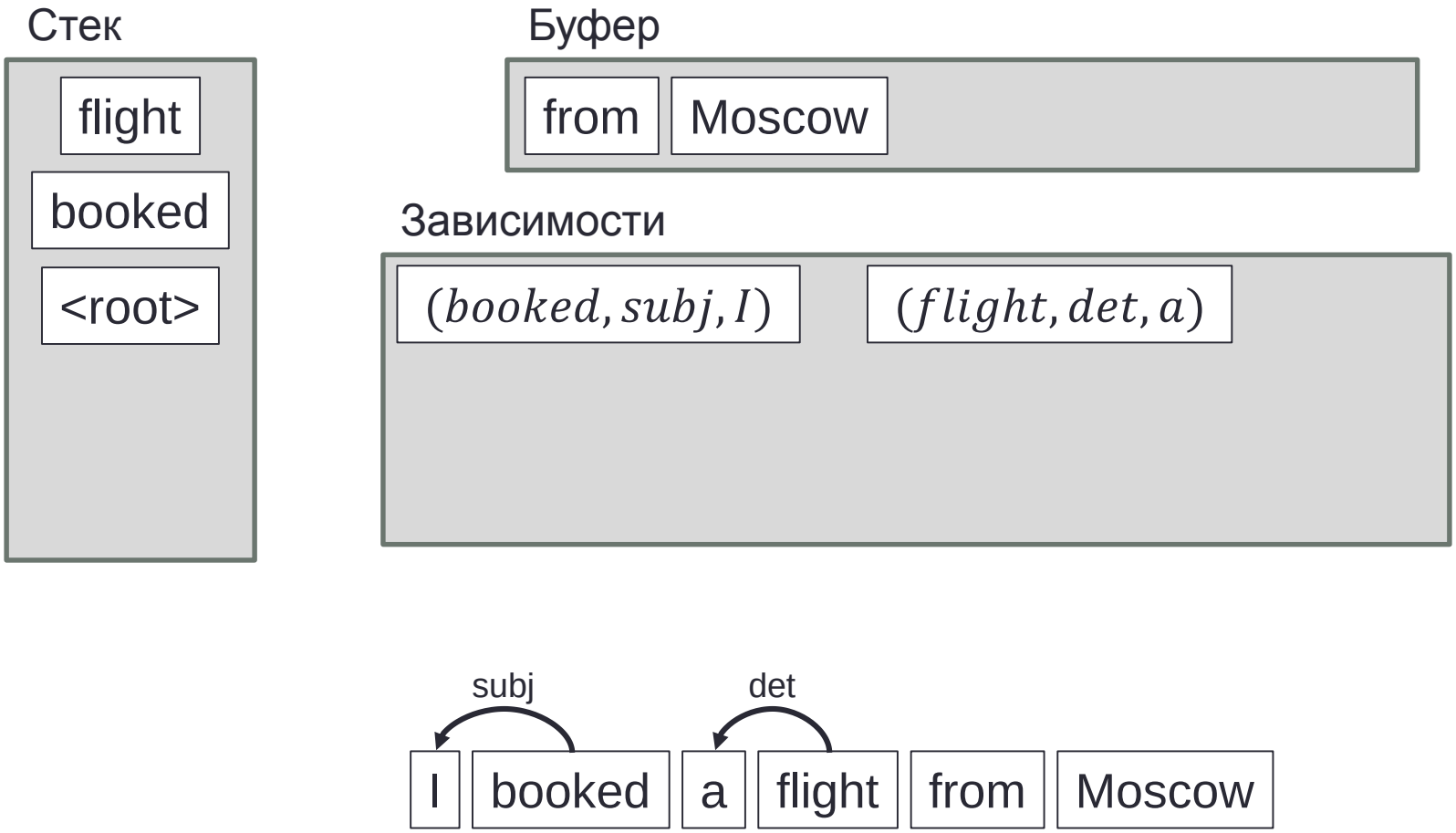
Стек



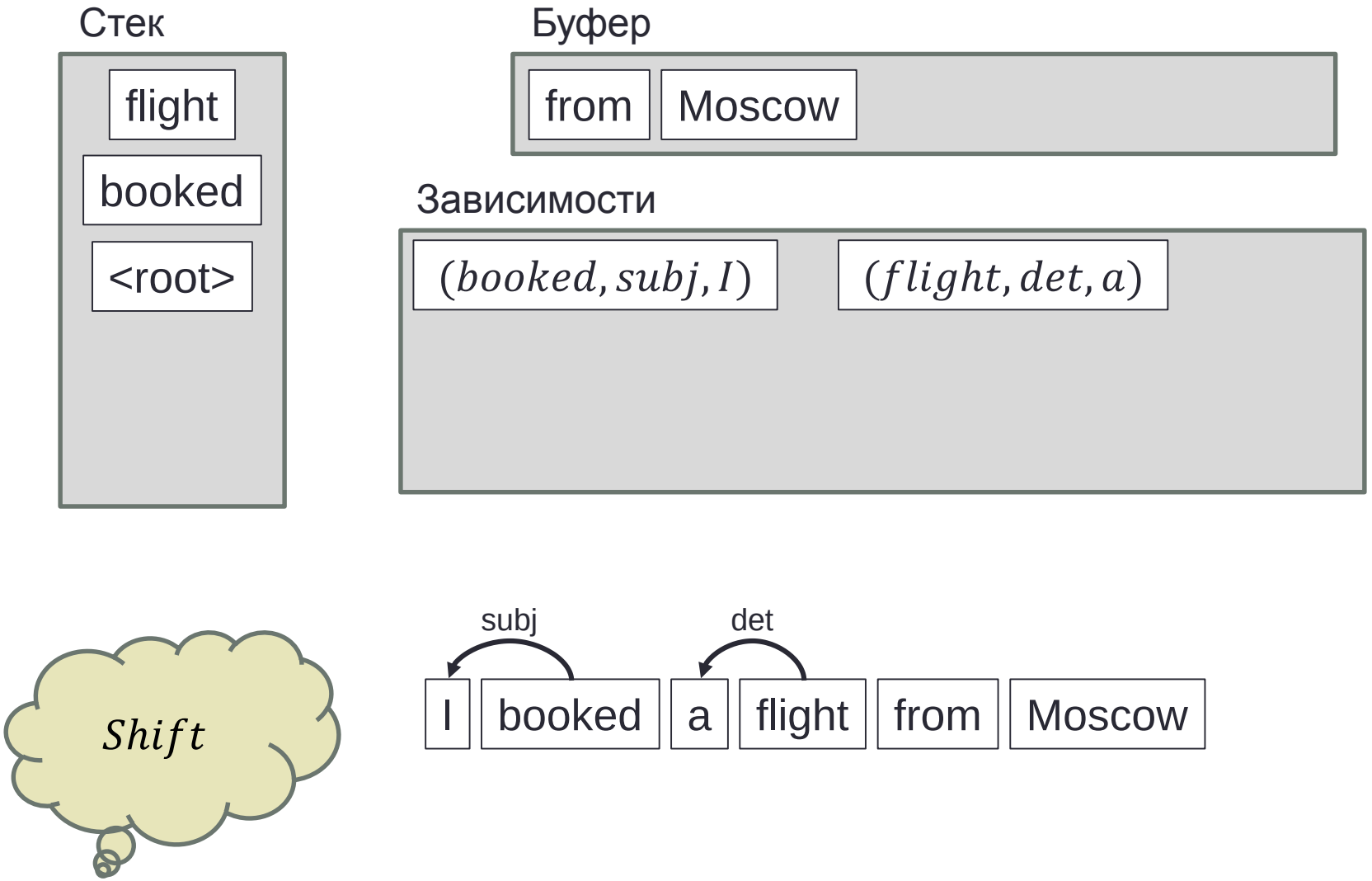
Буфер

Зависимости

Arc-standard



Arc-standard



Arc-standard

Стек

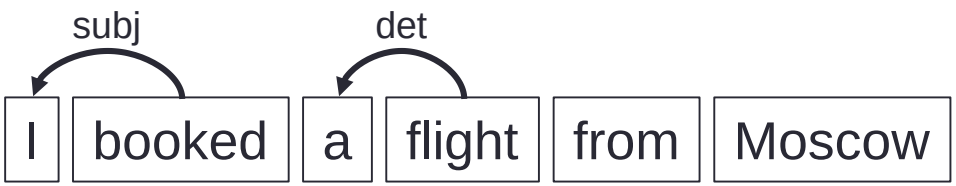
- from
- flight
- booked
- <root>

Буфер

- Moscow

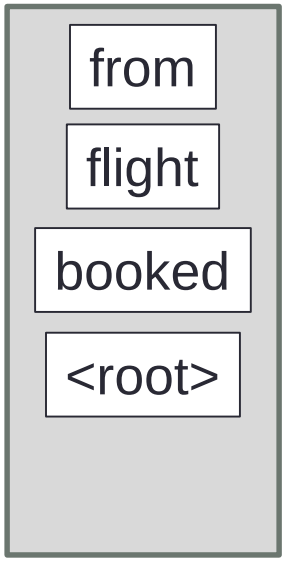
Зависимости

- $(booked, subj, I)$
- $(flight, det, a)$



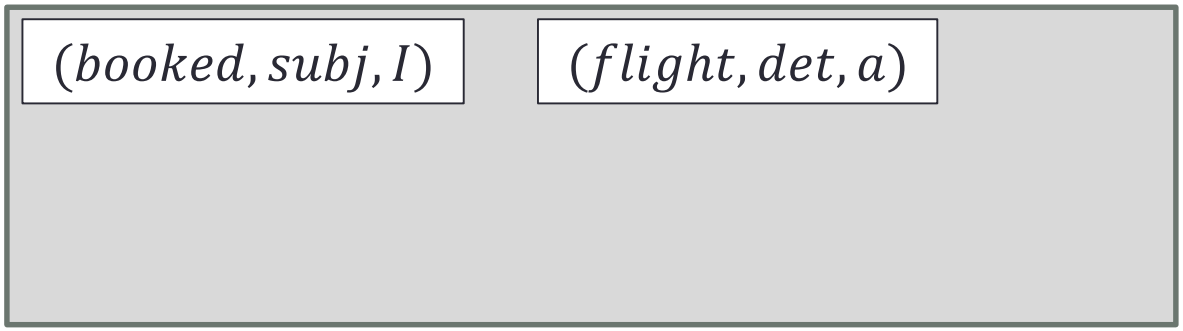
Arc-standard

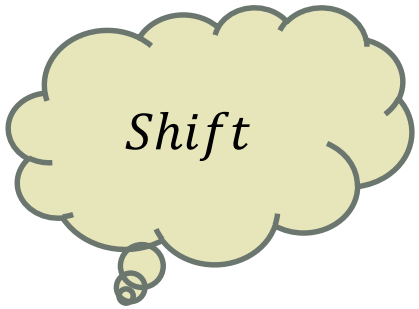
Стек

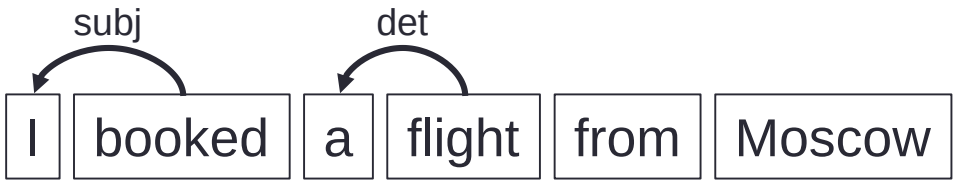


Буфер

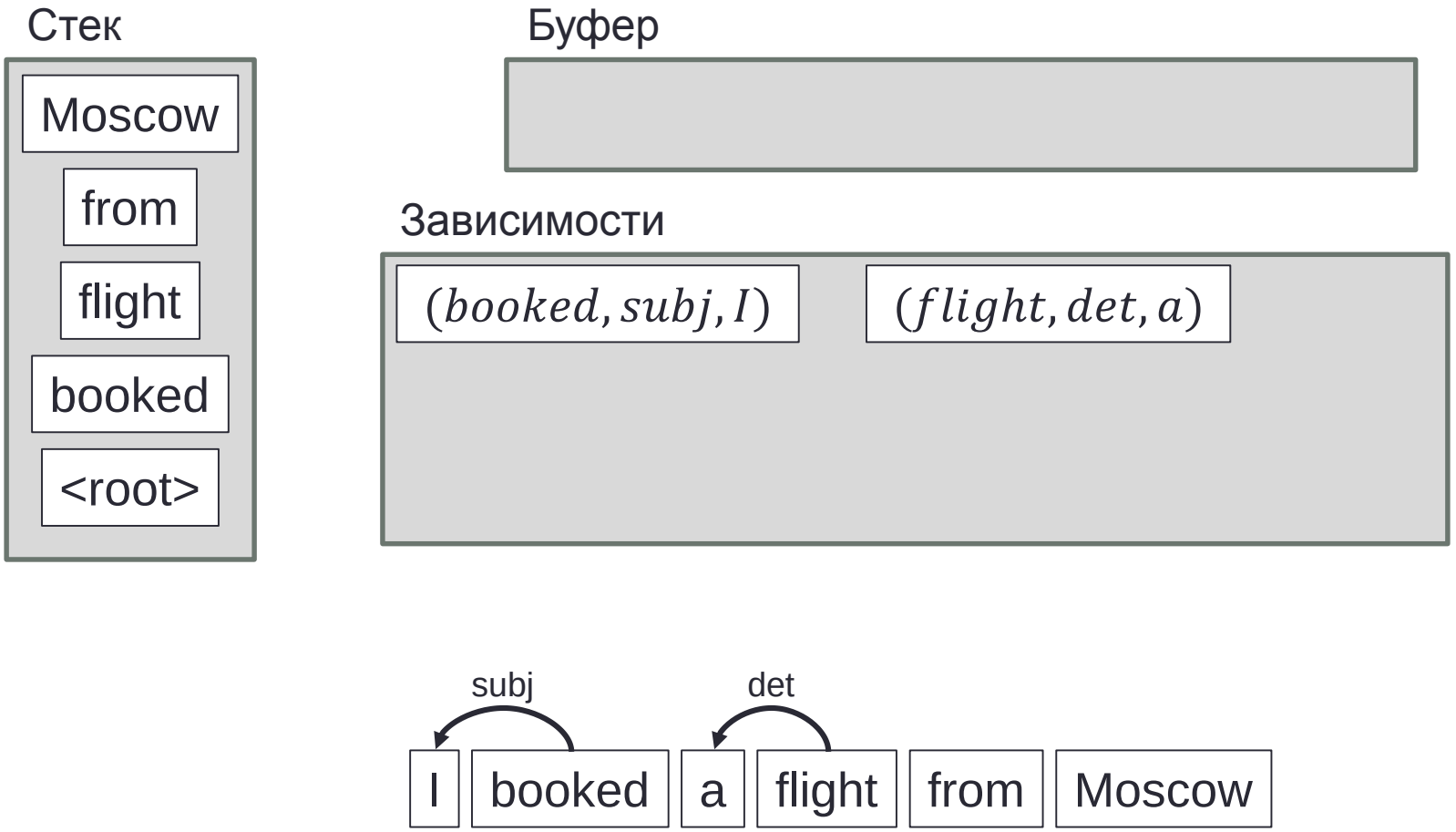
Зависимости





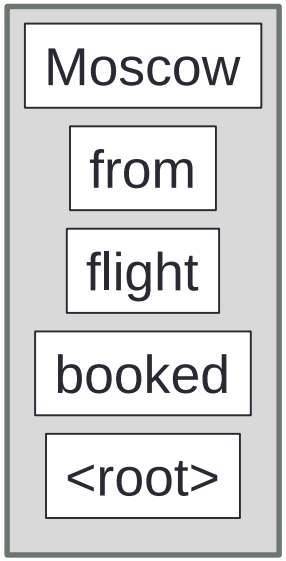


Arc-standard



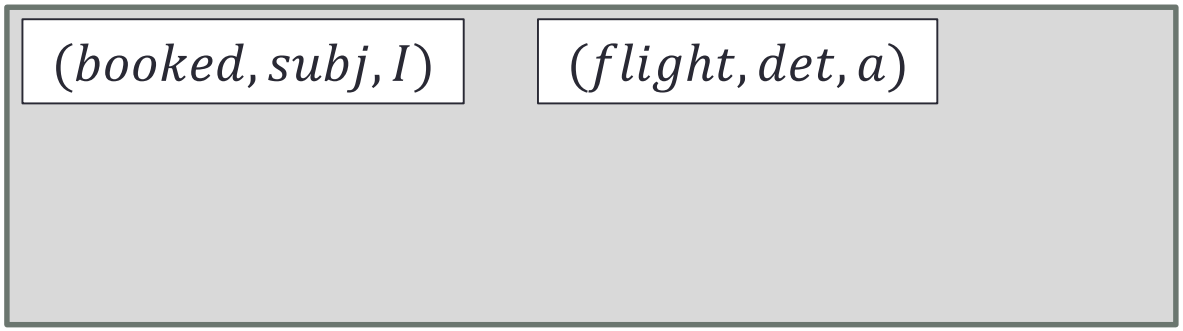
Arc-standard

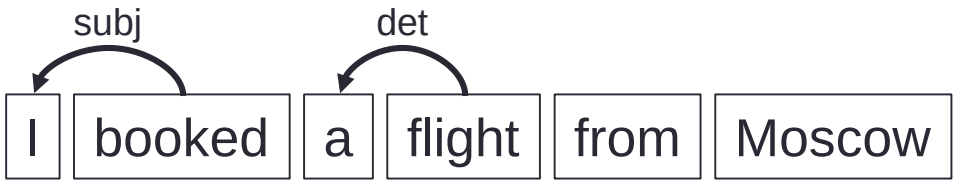
Стек



Буфер

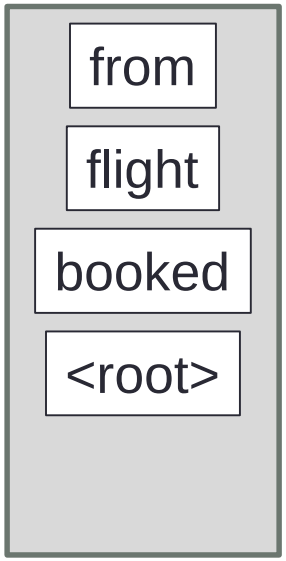
Зависимости





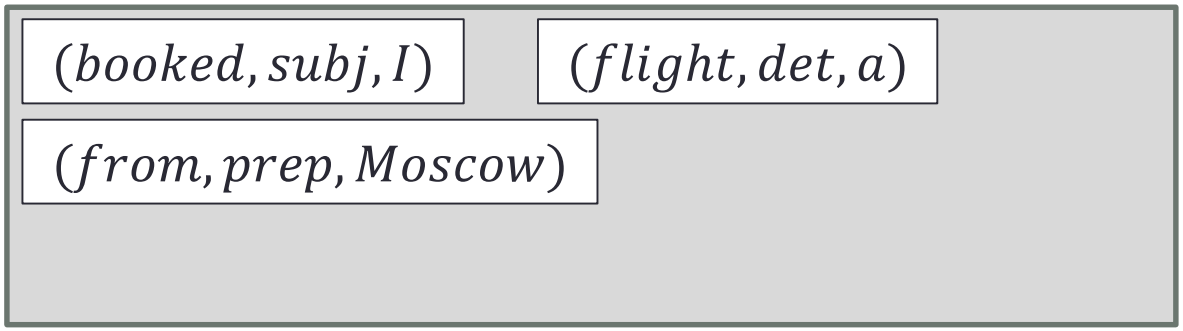
Arc-standard

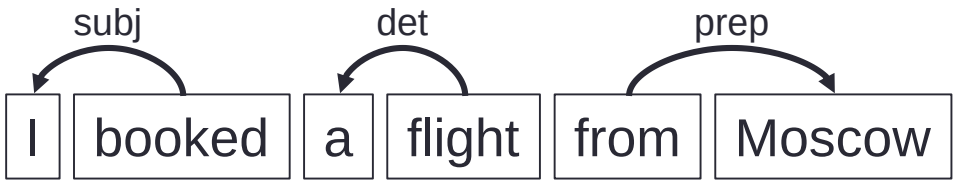
Стек



Буфер

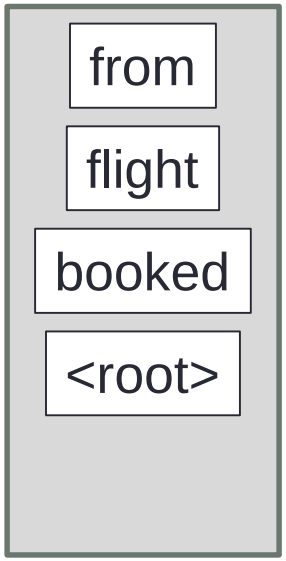
Зависимости





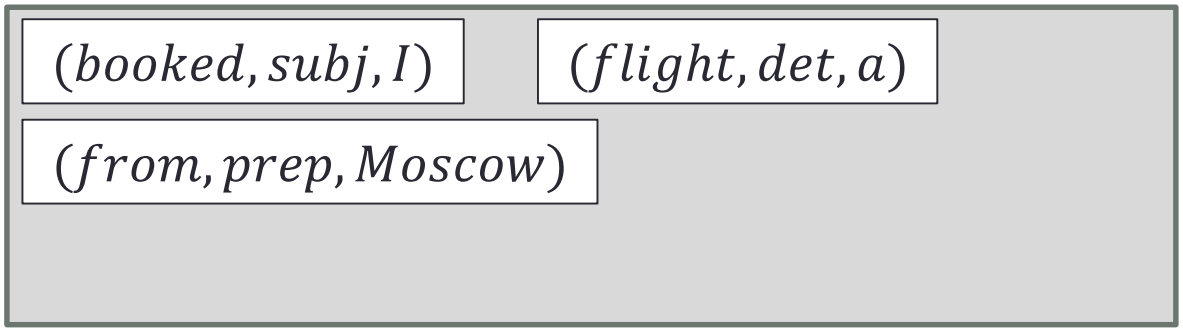
Arc-standard

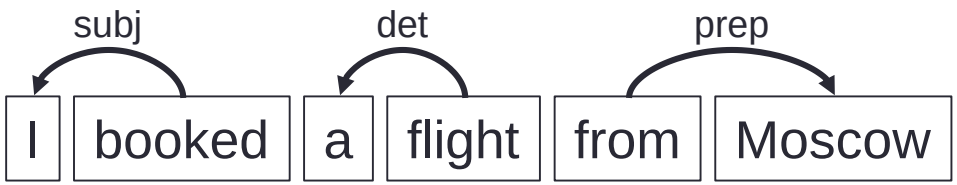
Стек



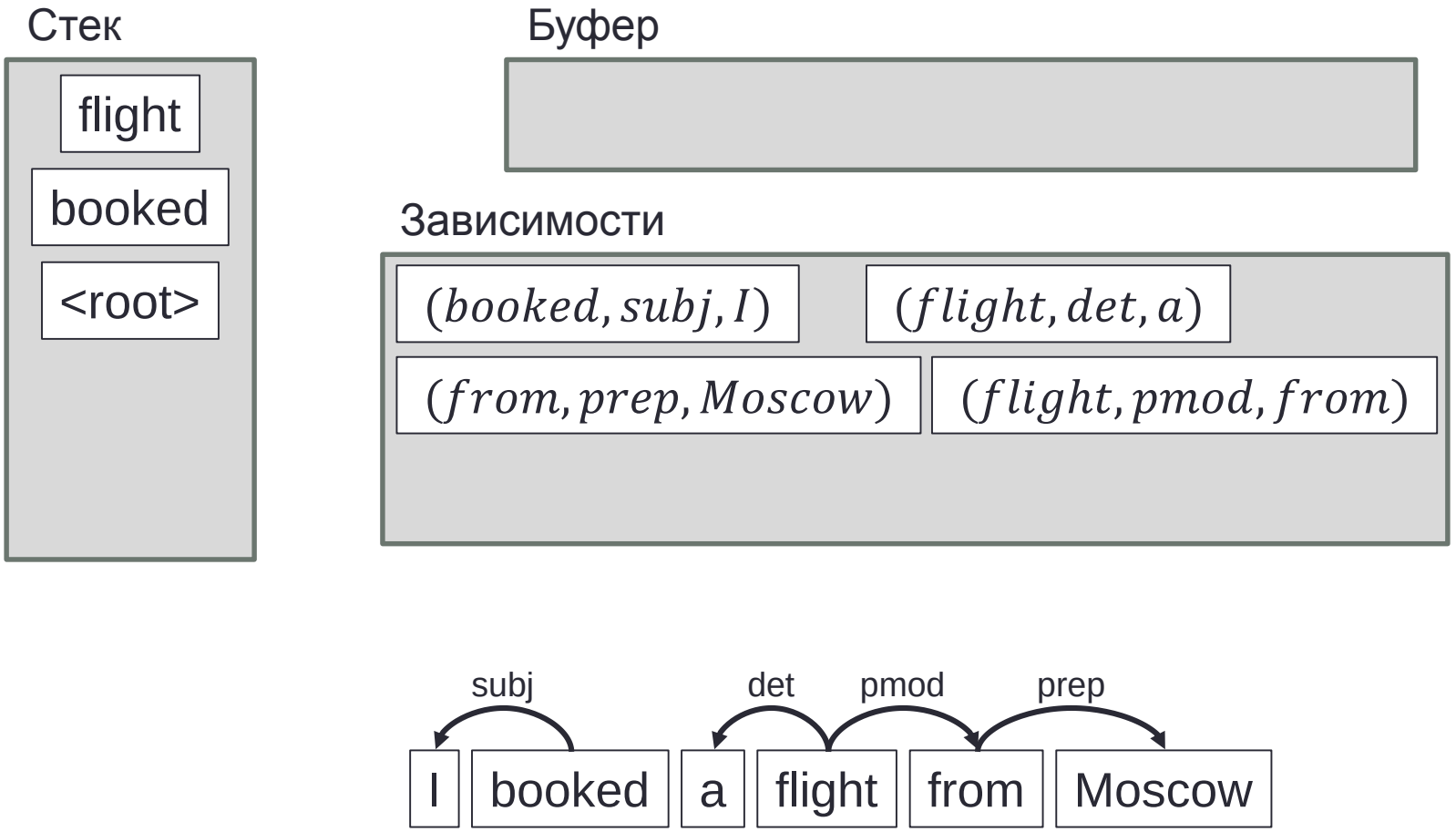
Буфер

Зависимости



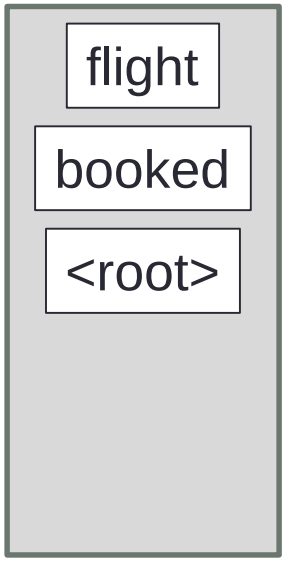


Arc-standard



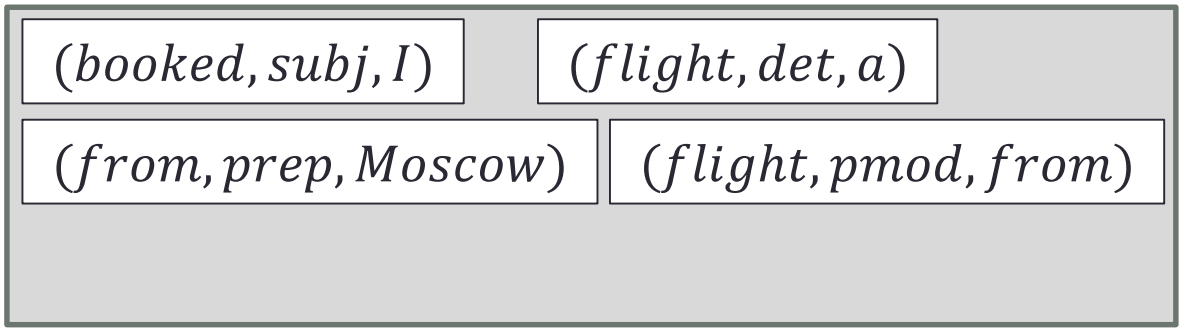
Arc-standard

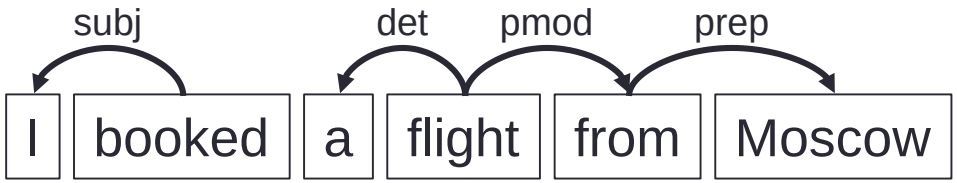
Стек



Буфер

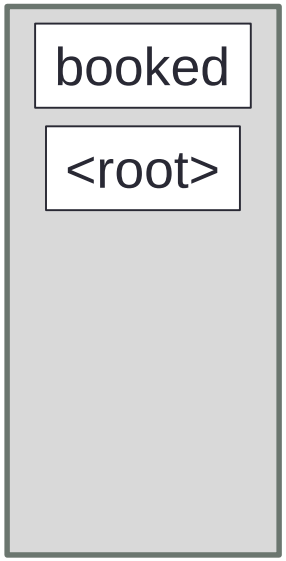
Зависимости





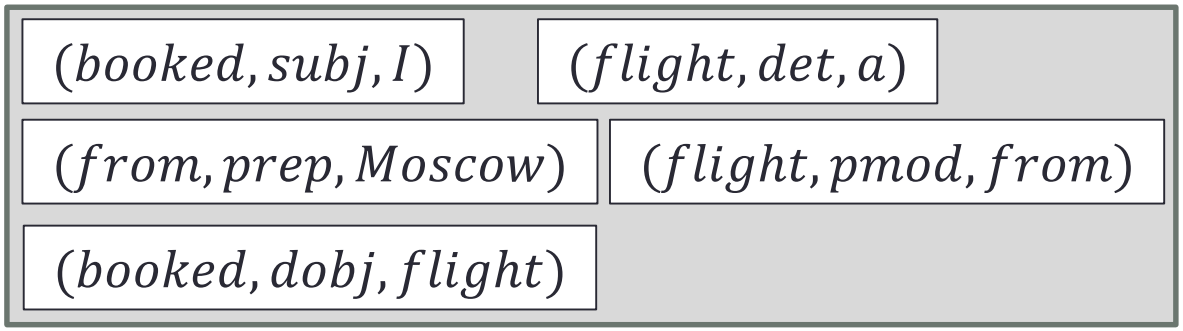
Arc-standard

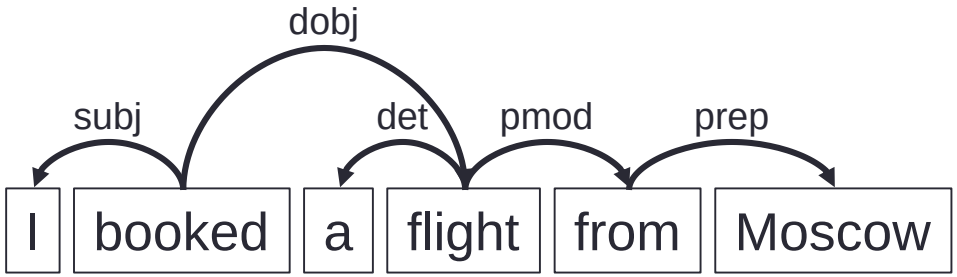
Стек



Буфер

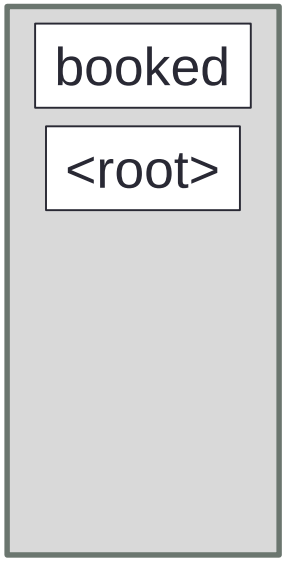
Зависимости





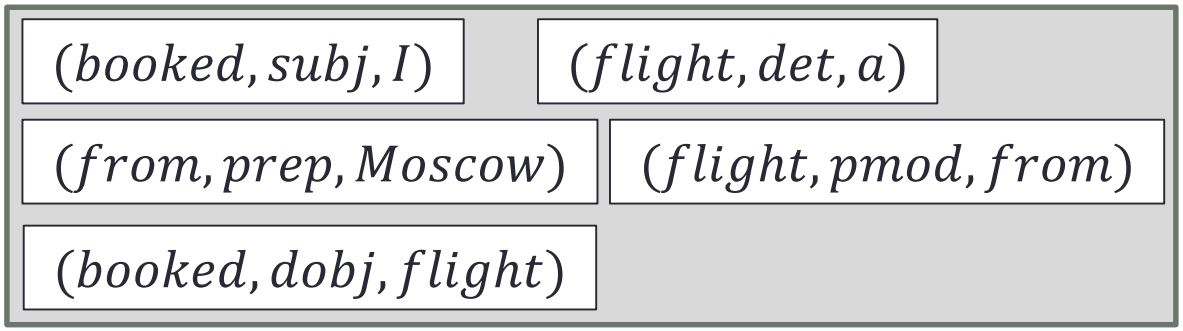
Arc-standard

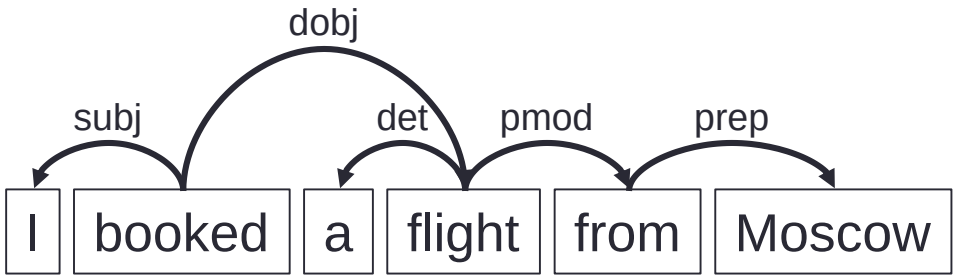
Стек



Буфер

Зависимости





Arc-standard

Стек

<root>

Буфер

Зависимости

$(booked, subj, I)$

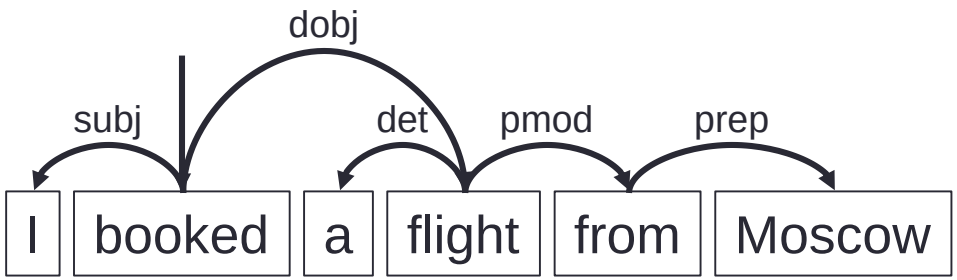
$(flight, det, a)$

$(from, prep, Moscow)$

$(flight, pmod, from)$

$(booked, dobj, flight)$

$(root, root, booked)$



Arc-standard

Стек

<root>

Буфер

Зависимости

$(booked, subj, I)$

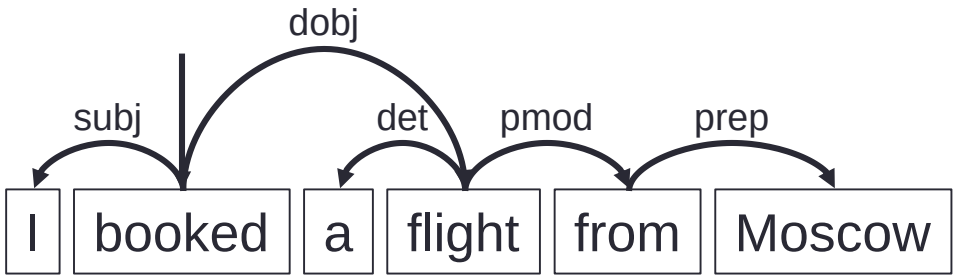
$(flight, det, a)$

$(from, prep, Moscow)$

$(flight, pmod, from)$

$(booked, dobj, flight)$

$(root, root, booked)$

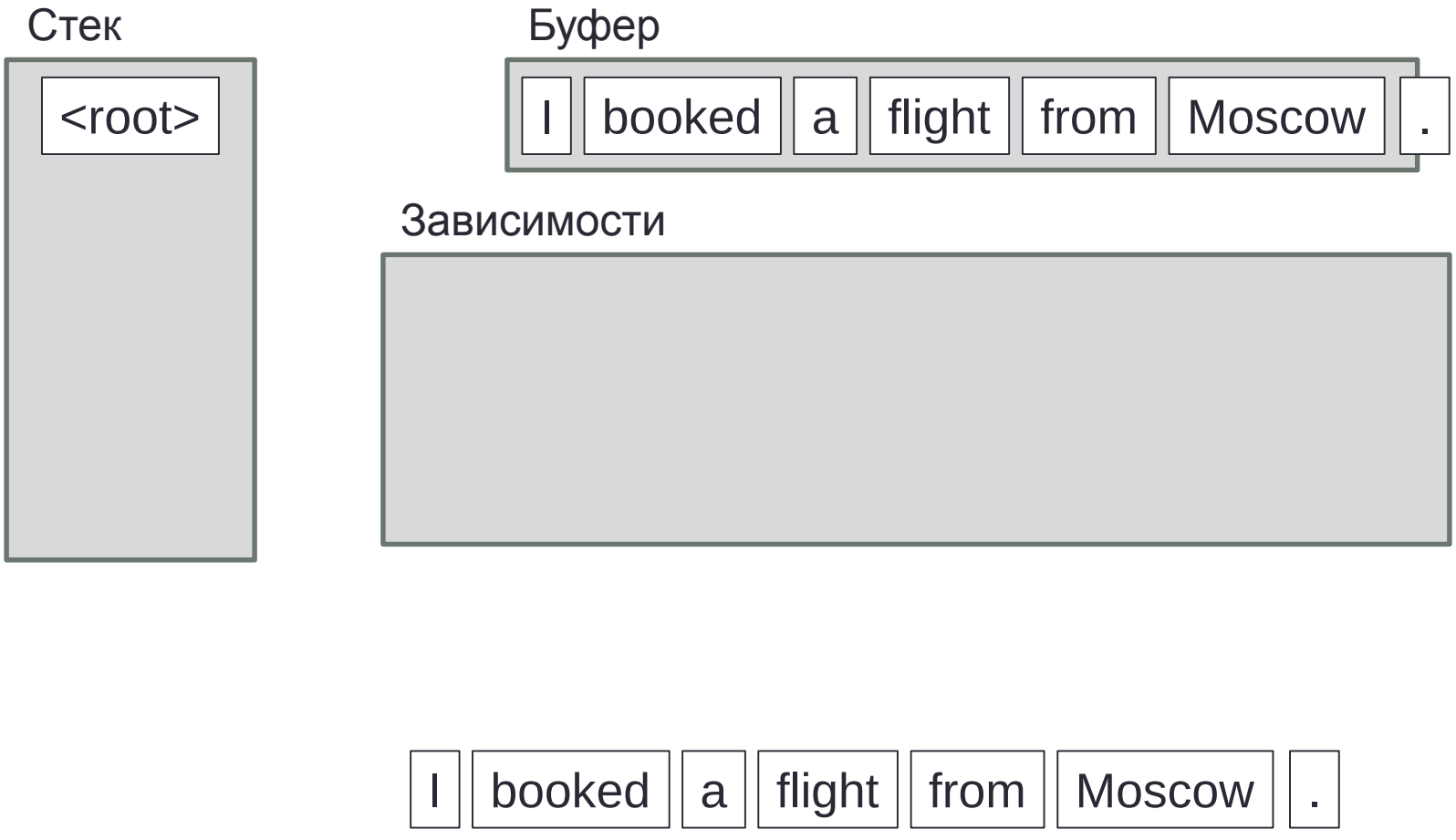


$DONE$!!!

Arc-eager

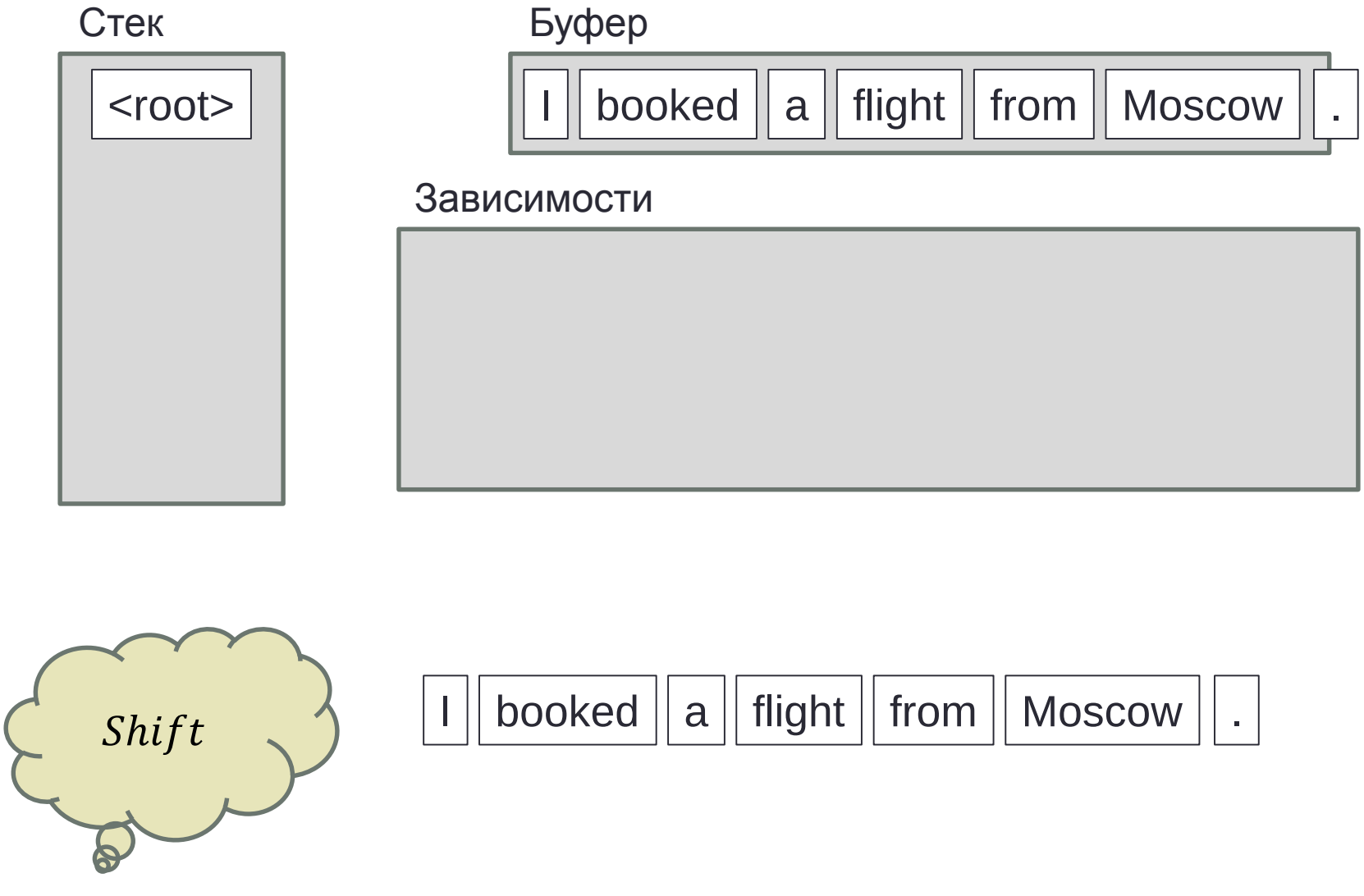
- Инициализация
  - $c_s(x = x_1 \dots x_n) = ([0], [1, \dots, n], \emptyset)$
- Конечное состояние
  - $C_t = \{c \in C | c = (\Sigma, [\,], A)\}$
- Переходы
  - $(Shift)$ $(\sigma, [i|\beta], A) \rightarrow ([\sigma|i], \beta, A)$
  - $(LeftArc_l)$ $([\sigma|i], [j|\beta], A) \rightarrow (\sigma, [j|\beta], A \cup \{(j, l, i)\})$, если $i \neq 0$ и $\nexists k$: $(k, *, i) \in A$
  - $(RightArc_l)$ $([\sigma|i], [j|\beta], A) \rightarrow ([\sigma|i|j], \beta, A \cup \{(i, l, j)\})$
  - $(Reduce)$ $([\sigma|i], B, A) \rightarrow (\sigma, B, A)$ если $\exists k$: $(k, *, i) \in A$

Arc-eager

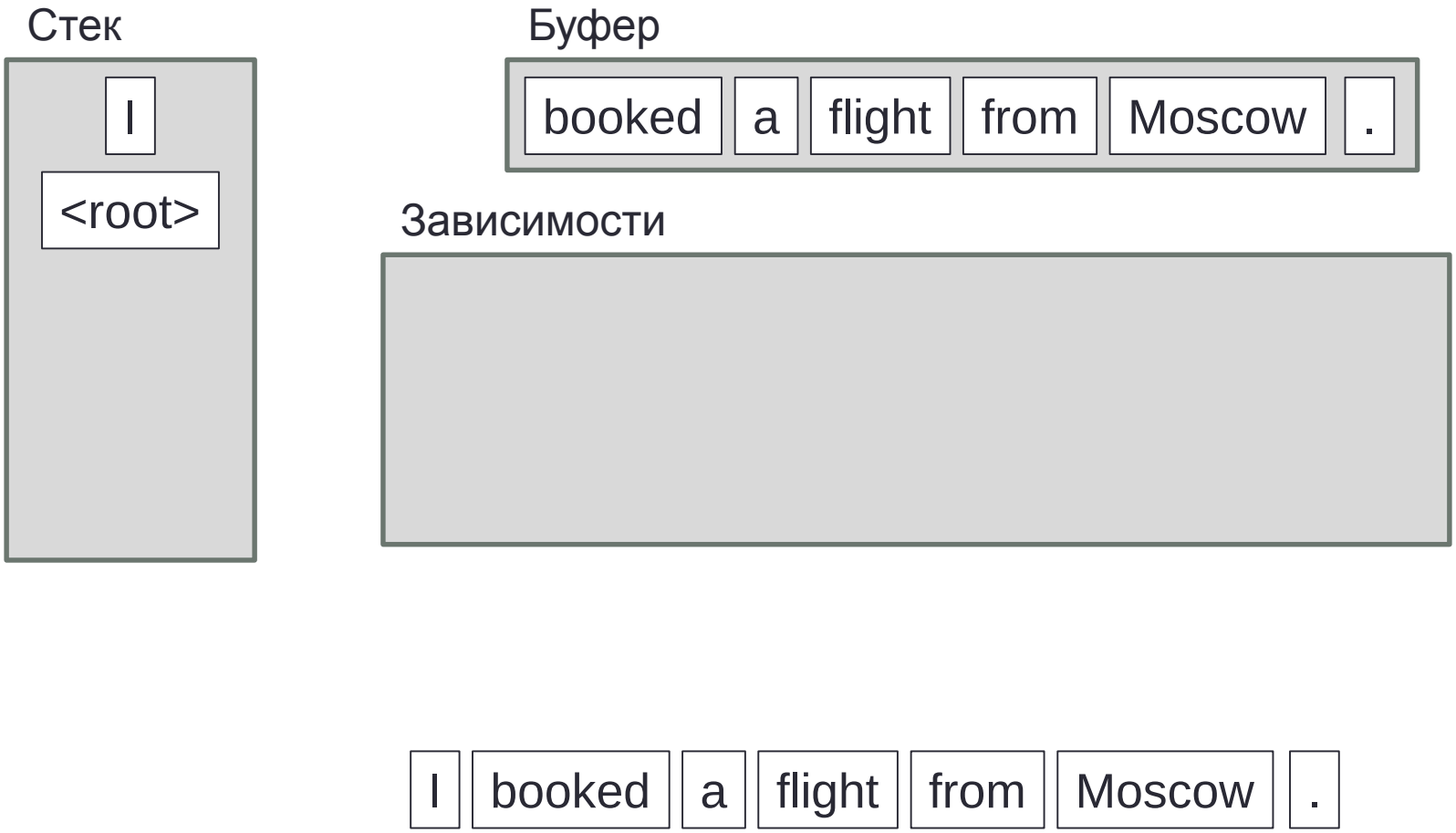


I | booked | a | flight | from | Moscow | .

Arc-eager

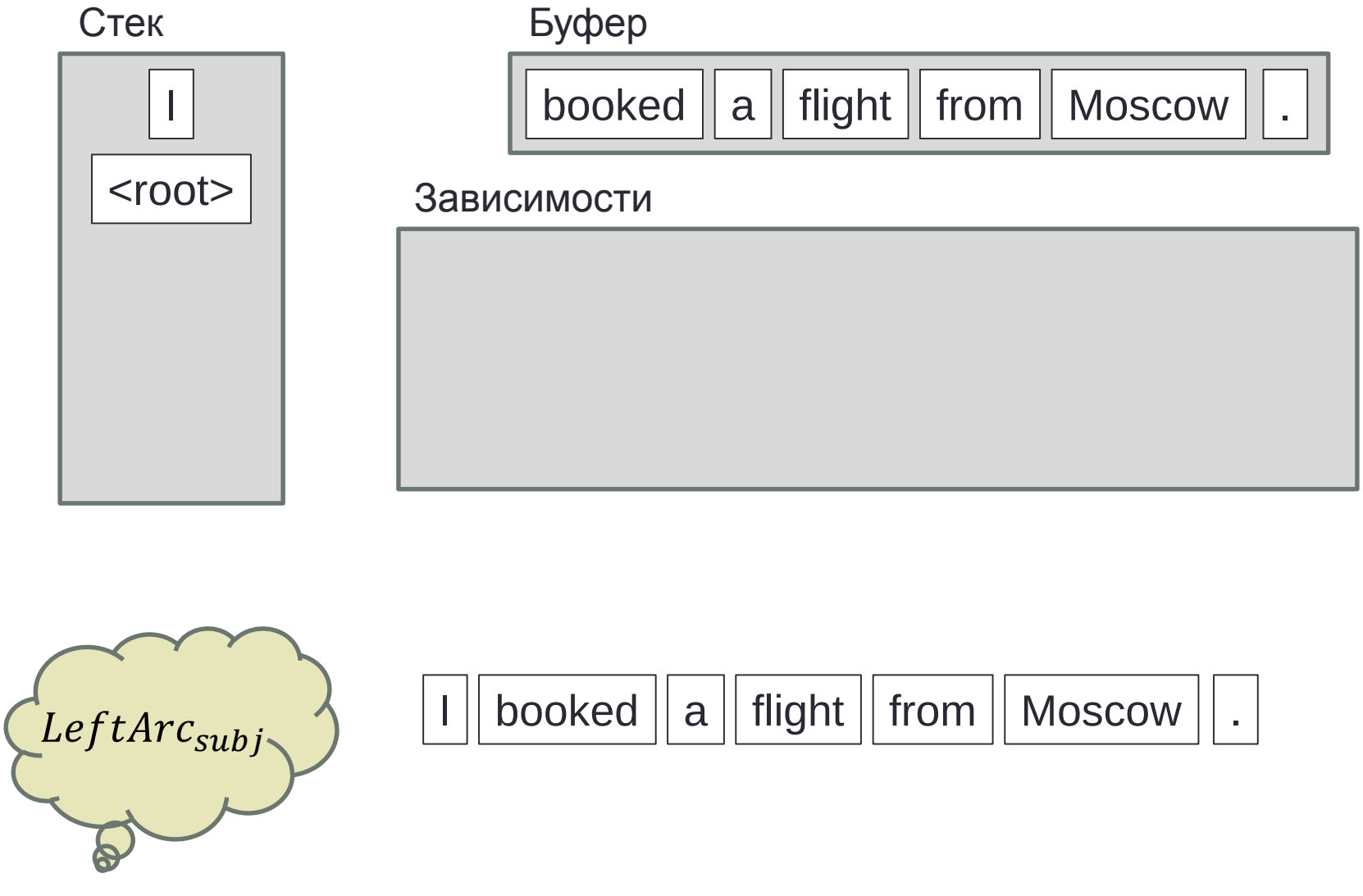

Стек
<root>
Буфер
I
booked
a
flight
from
Moscow
.
Зависимости
Shift
I
booked
a
flight
from
Moscow
.

Arc-eager

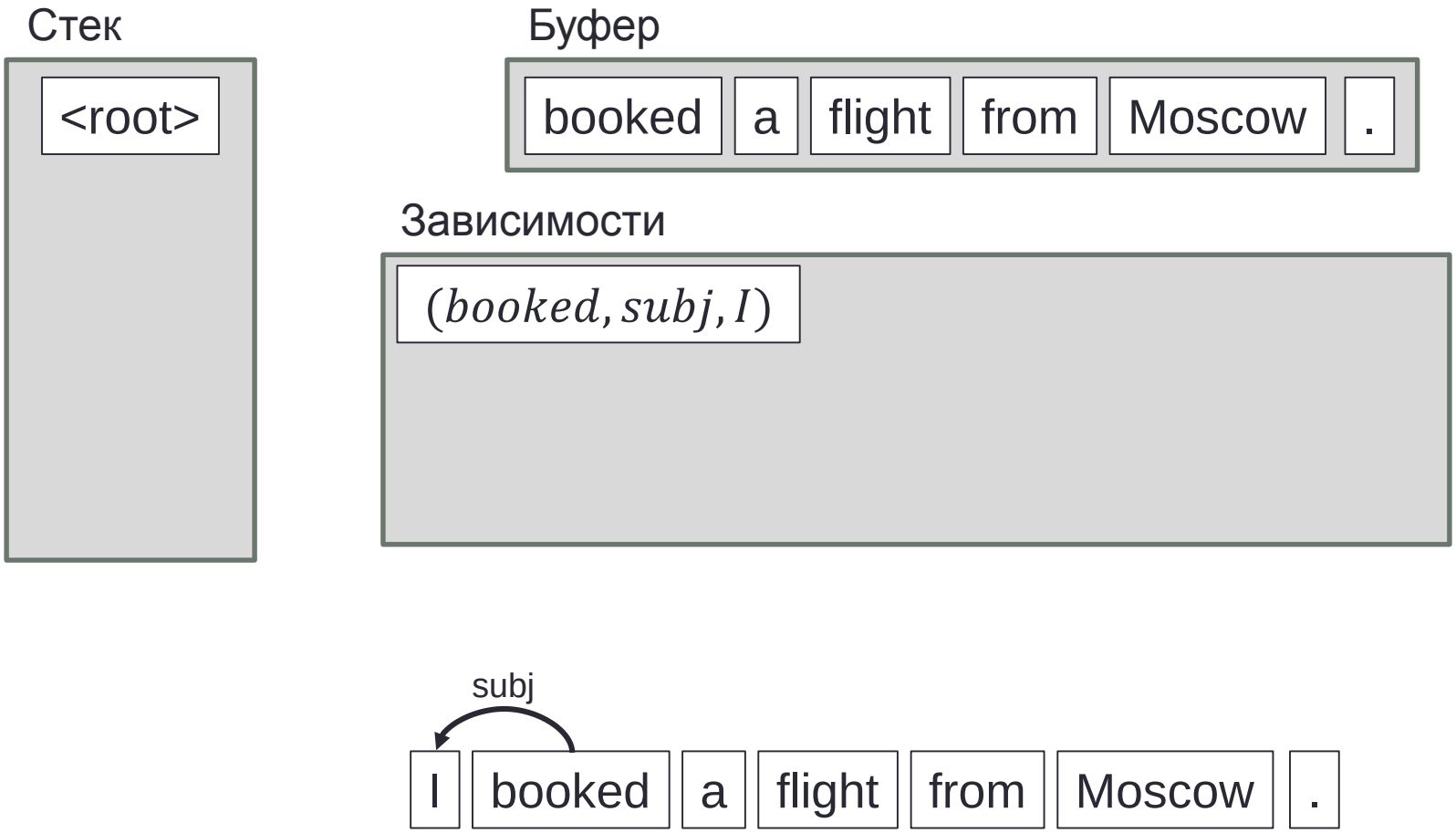


I | booked | a | flight | from | Moscow | .

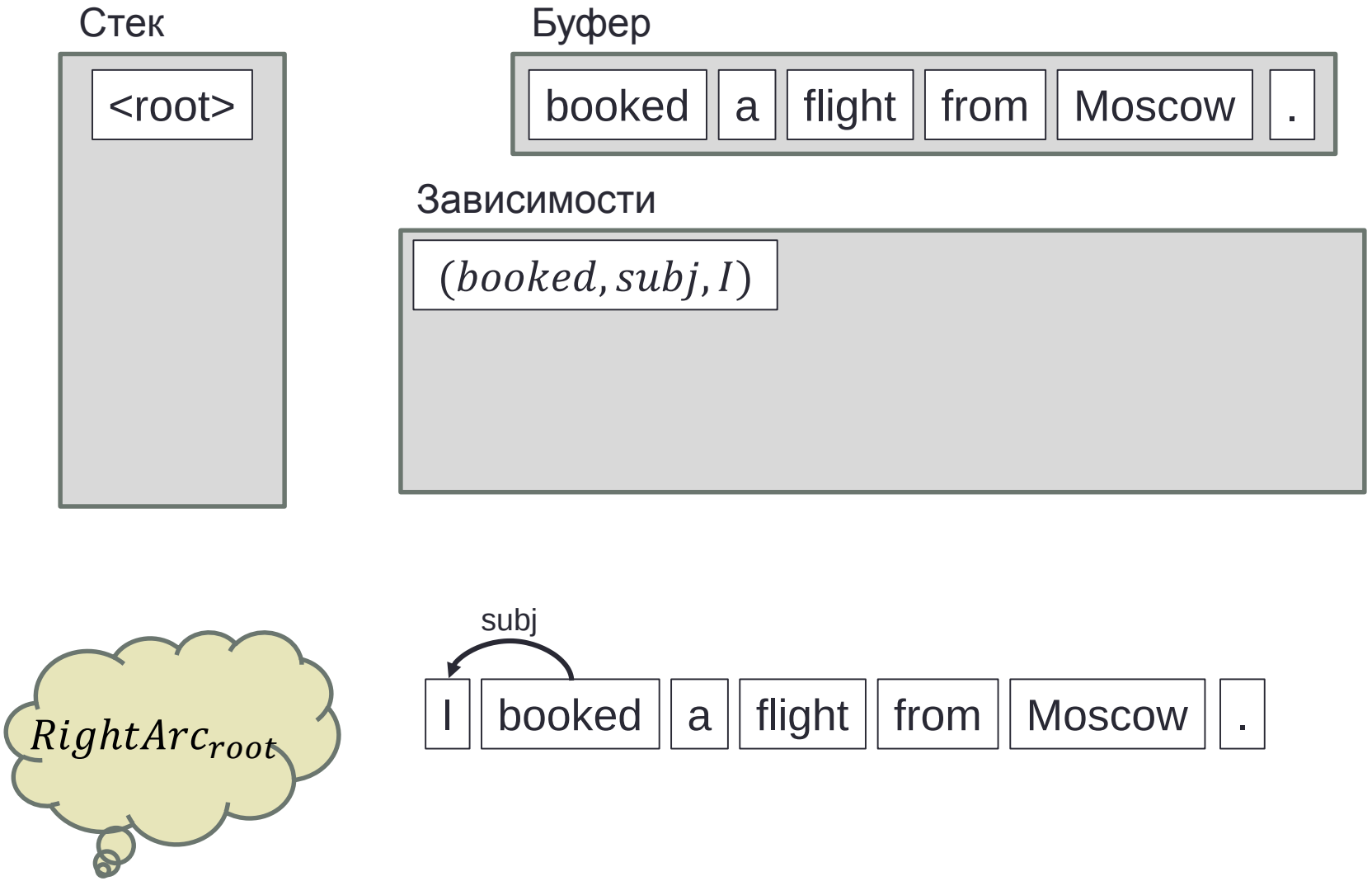
Arc-eager


Стек
I
<root>
Буфер
booked
a
flight
from
Moscow
.
Зависимости
$LeftArc_{subj}$
I
booked
a
flight
from
Moscow
.

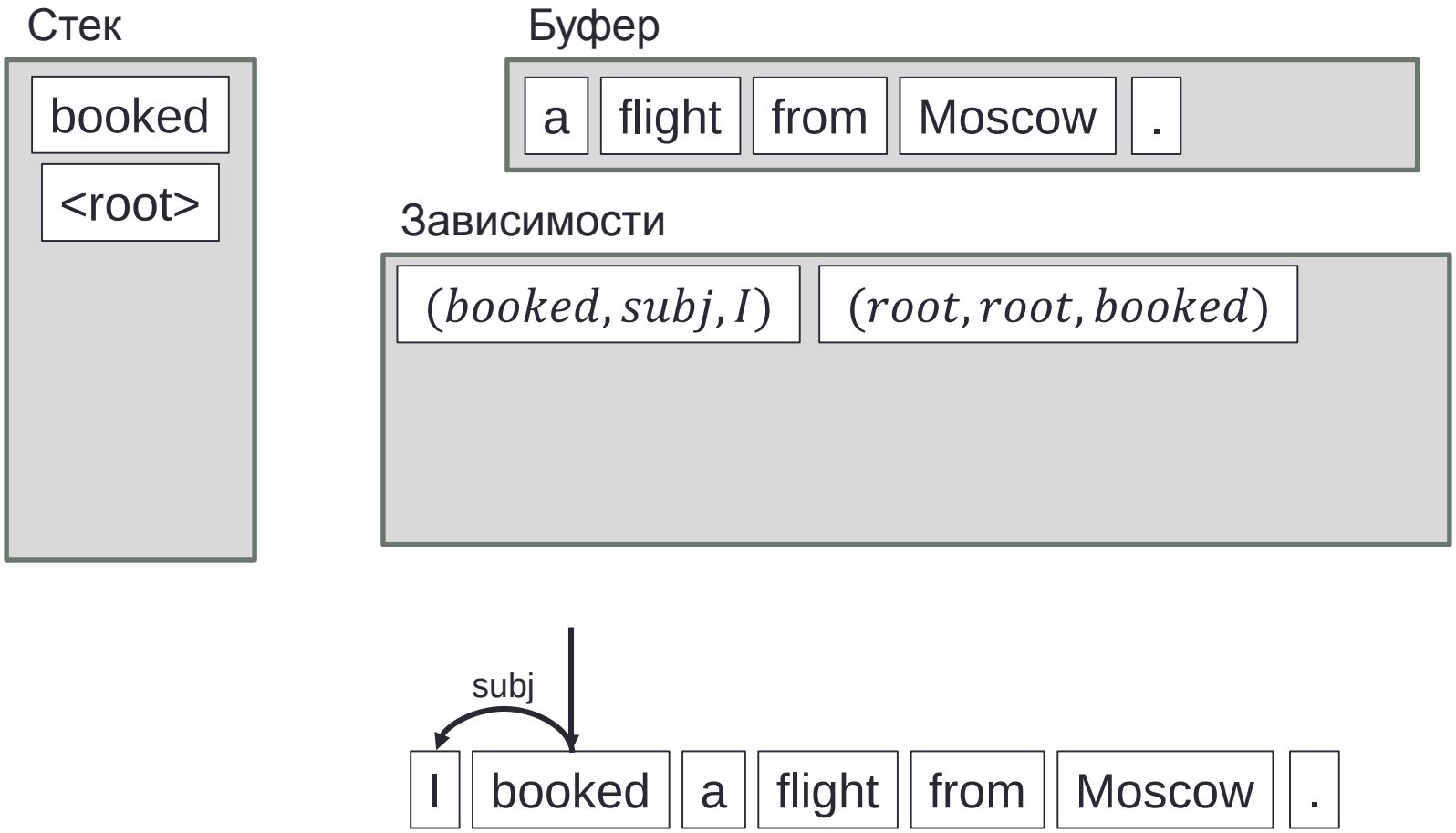
Arc-eager


Стек
<root>
Буфер
booked
a
flight
from
Moscow
.
Зависимости
$(booked, subj, I)$
subj
I
booked
a
flight
from
Moscow
.

Arc-eager

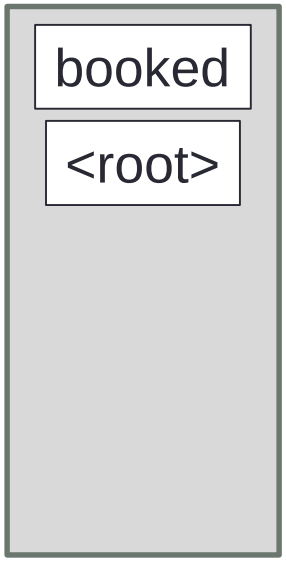

Стек
<root>
Буфер
booked
a
flight
from
Moscow
.
Зависимости
$(booked, subj, I)$
$RightArc_{root}$
subj
I
booked
a
flight
from
Moscow
.

Arc-eager


Стек
booked
<root>
Буфер
a
flight
from
Moscow
.
Зависимости
$(booked, subj, I)$
$(root, root, booked)$
subj
I
booked
a
flight
from
Moscow
.

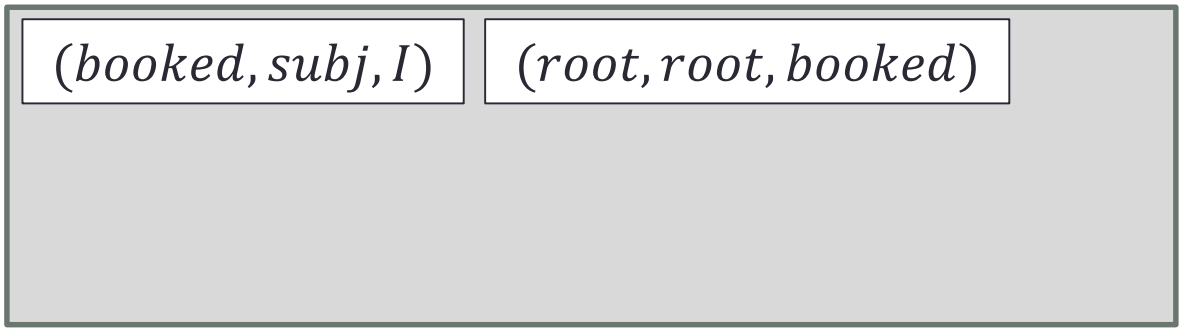
Arc-eager

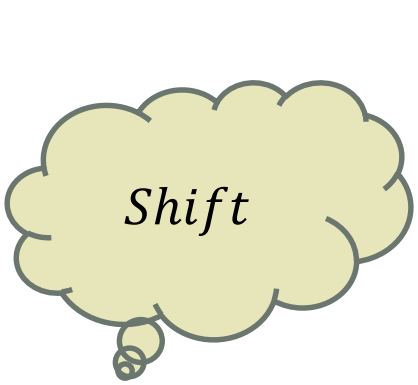
Стек



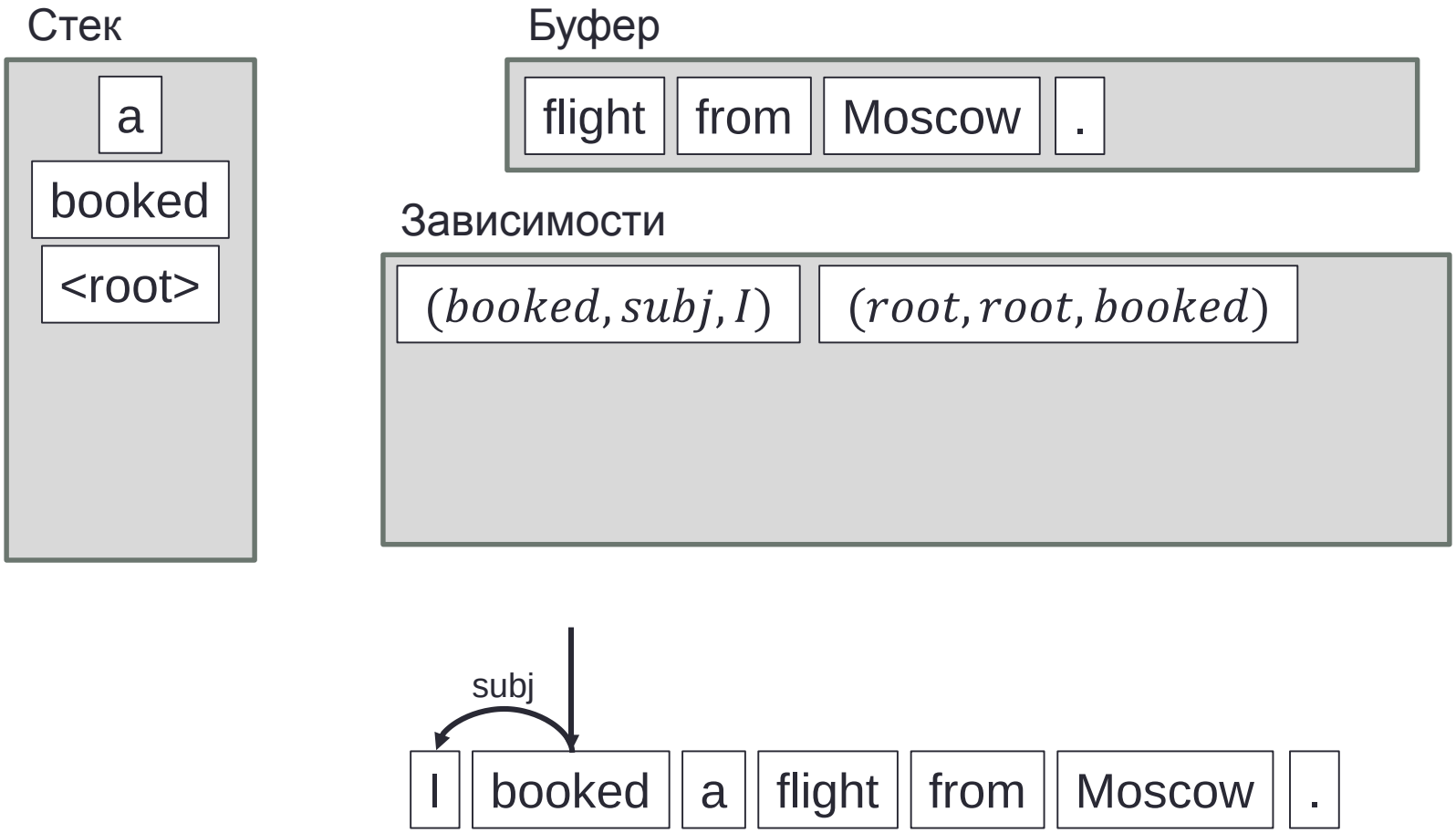
Буфер

Зависимости

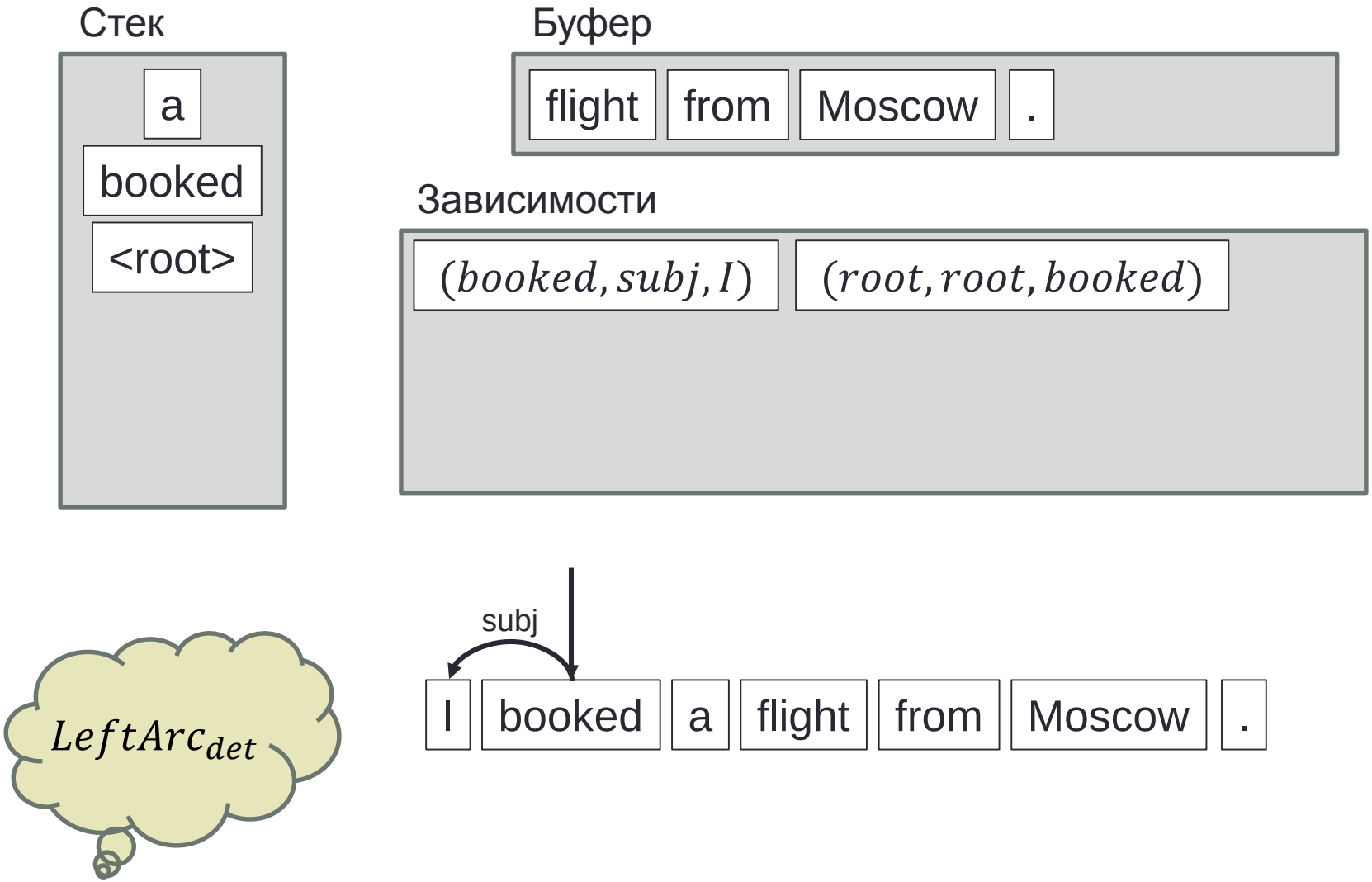




Arc-eager

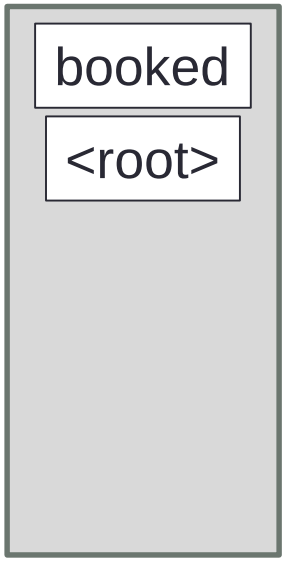

Стек
a
booked
<root>
Буфер
flight
from
Moscow
.
Зависимости
$(booked, subj, I)$
$(root, root, booked)$
subj
I
booked
a
flight
from
Moscow
.

Arc-eager


Стек
a
booked
<root>
Буфер
flight
from
Moscow
.
Зависимости
$(booked, subj, I)$
$(root, root, booked)$
subj
I
booked
a
flight
from
Moscow
.
$LeftArc_{det}$

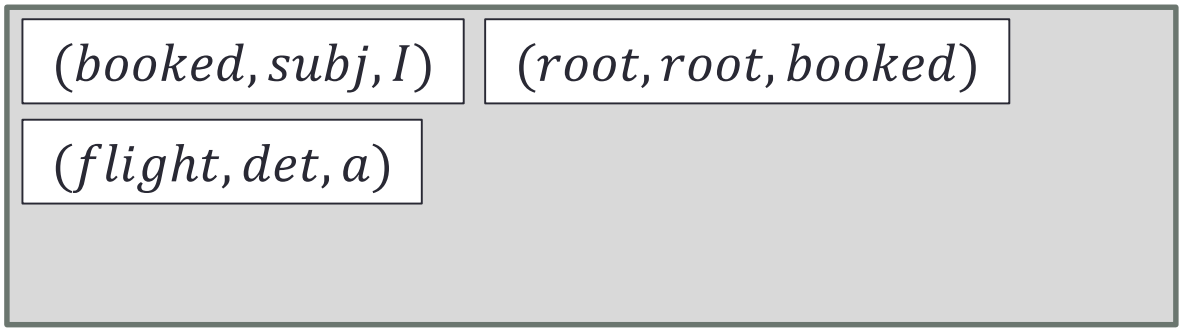
Arc-eager

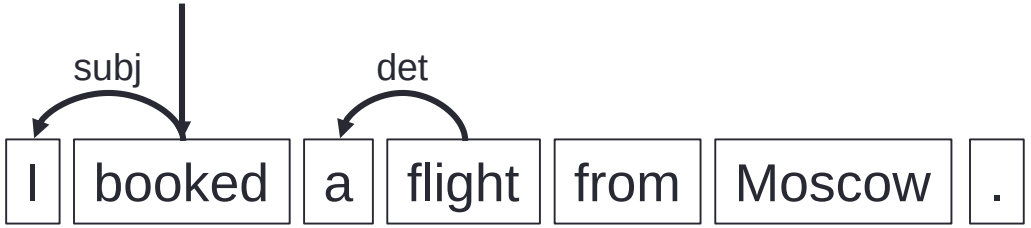
Стек



Буфер

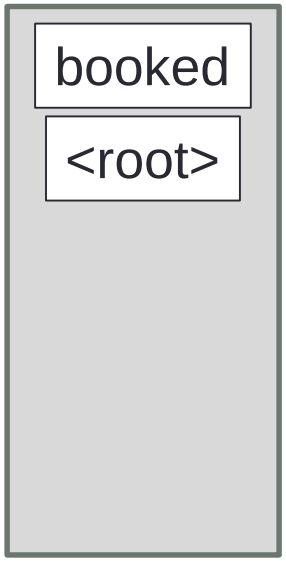
Зависимости





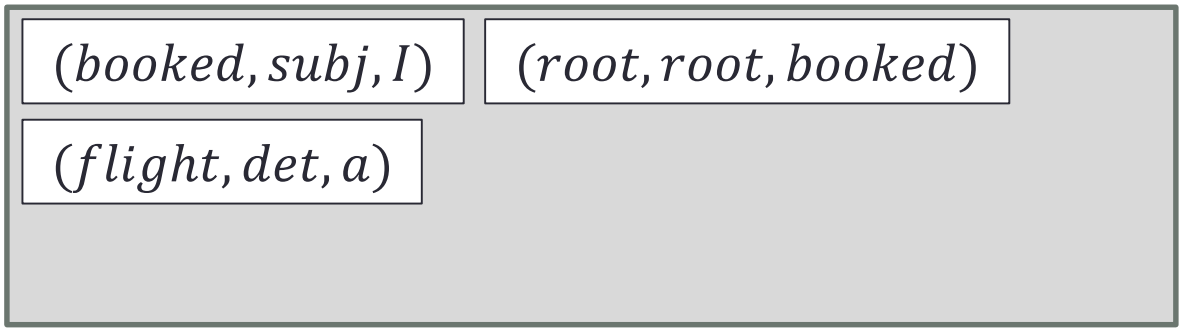
Arc-eager

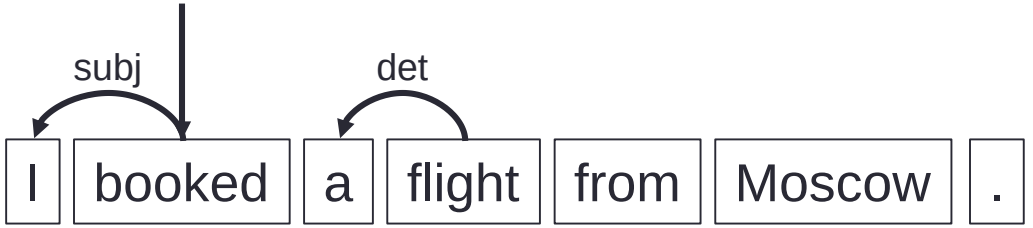
Стек



Буфер

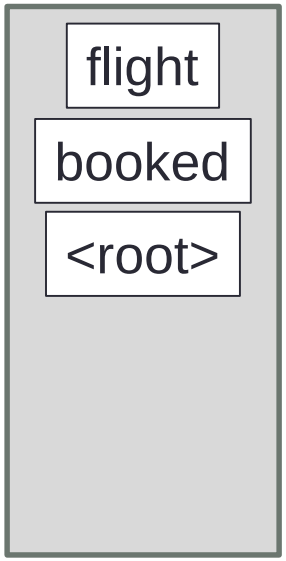
Зависимости



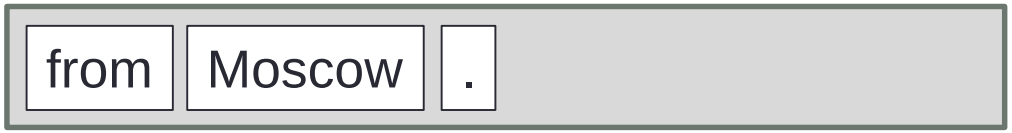


Arc-eager

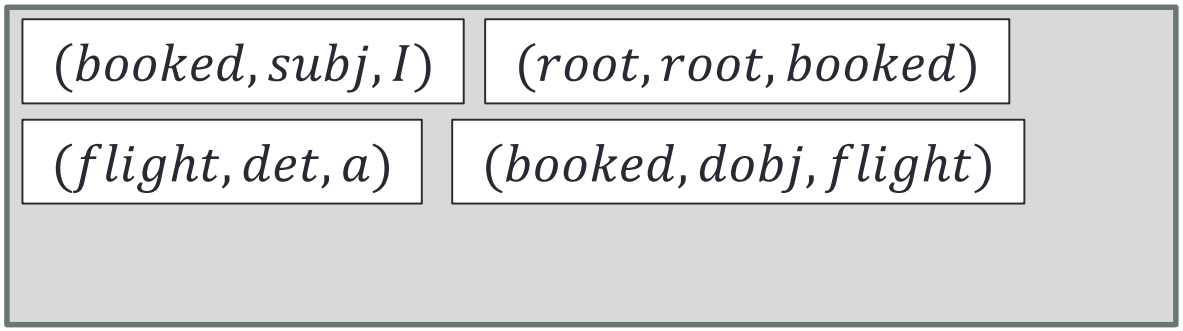
Стек

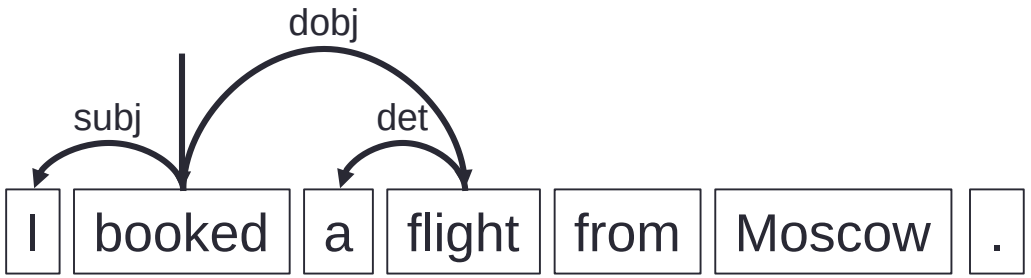


Буфер



Зависимости





Arc-eager

Стек

flight

booked

<root>

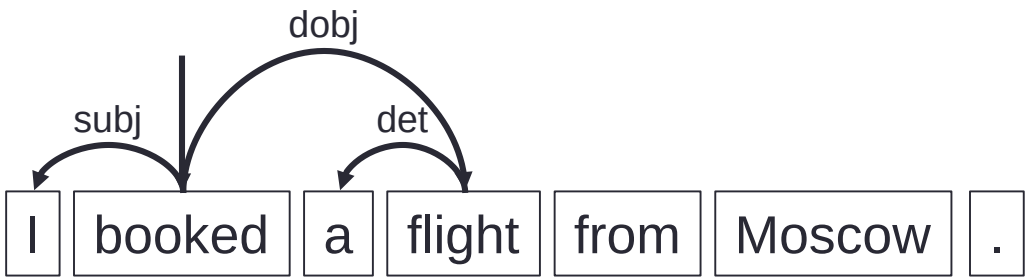
Буфер

from | Moscow | .

Зависимости

$(booked, subj, I)$ $(root, root, booked)$

$(flight, det, a)$ $(booked, dobj, flight)$



$RightArc_{pmod}$

Arc-eager

Стек

- from
- flight
- booked
- <root>

Буфер

Moscow .

Зависимости

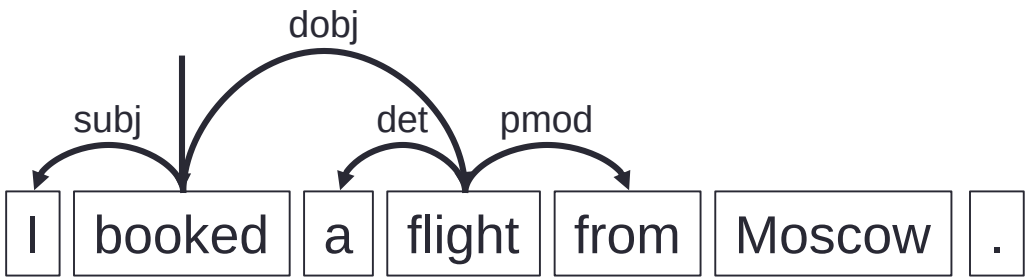
$(booked, subj, I)$

$(root, root, booked)$

$(flight, det, a)$

$(booked, dobj, flight)$

$(flight, pmod, from)$



Arc-eager

Стек

- from
- flight
- booked
- <root>

Буфер

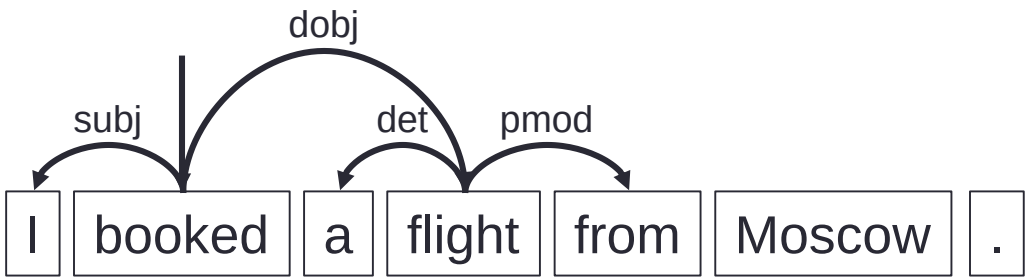
Moscow .

Зависимости

$(booked, subj, I)$ $(root, root, booked)$

$(flight, det, a)$ $(booked, dobj, flight)$

$(flight, pmod, from)$



$RightArc_{prep}$

Arc-eager

Стек

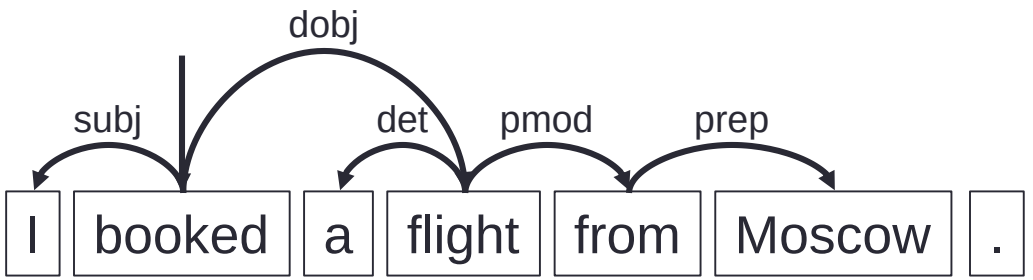
- Moscow
- from
- flight
- booked
- <root>

Буфер

- .

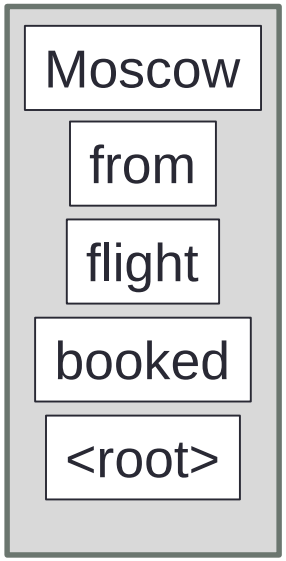
Зависимости

- $(booked, subj, I)$
- $(root, root, booked)$
- $(flight, det, a)$
- $(booked, dobj, flight)$
- $(flight, pmod, from)$
- $(from, prep, Moscow)$



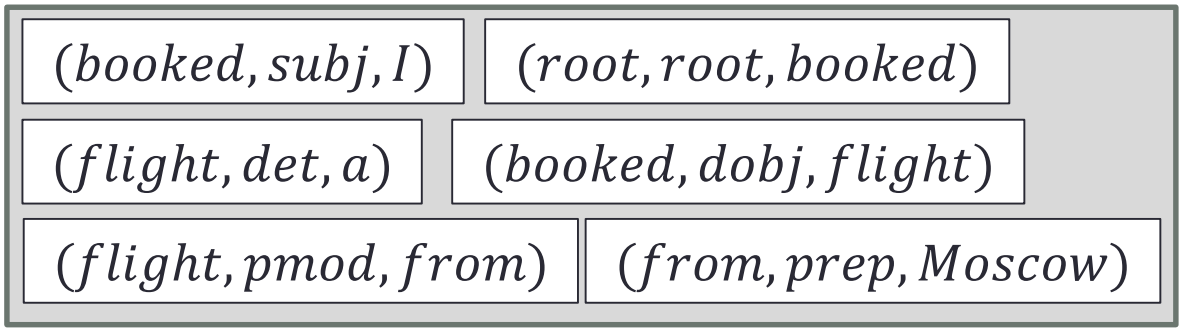
Arc-eager

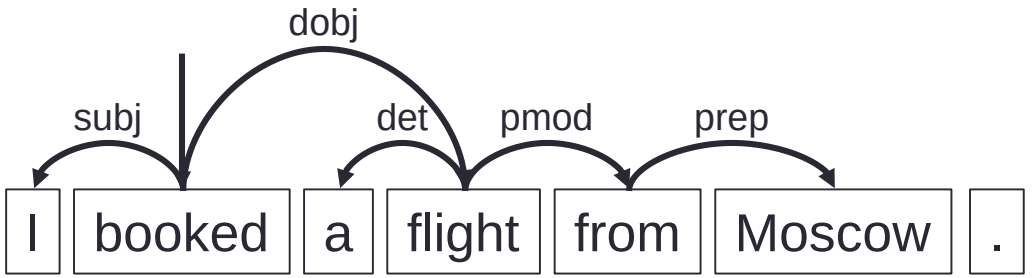
Стек



Буфер

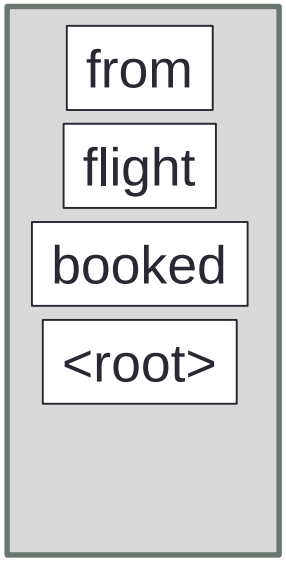
Зависимости





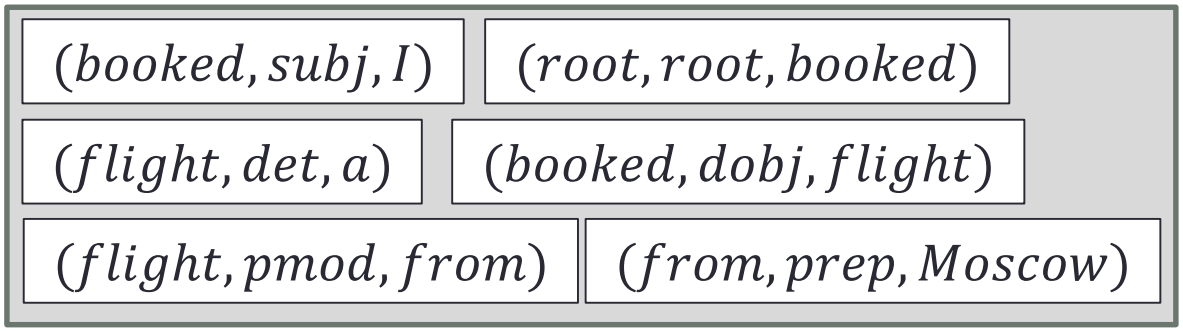
Arc-eager

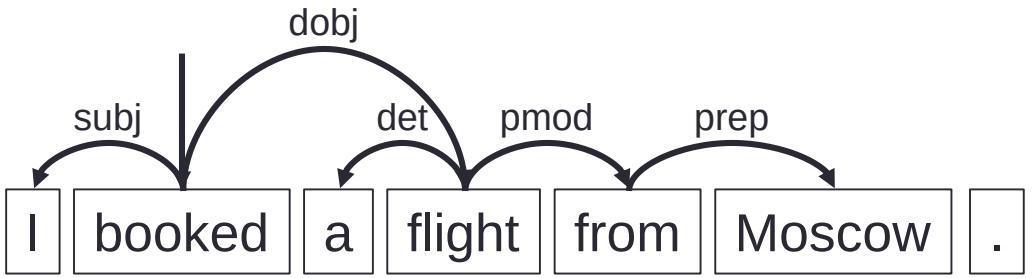
Стек



Буфер

Зависимости





Arc-eager

Стек

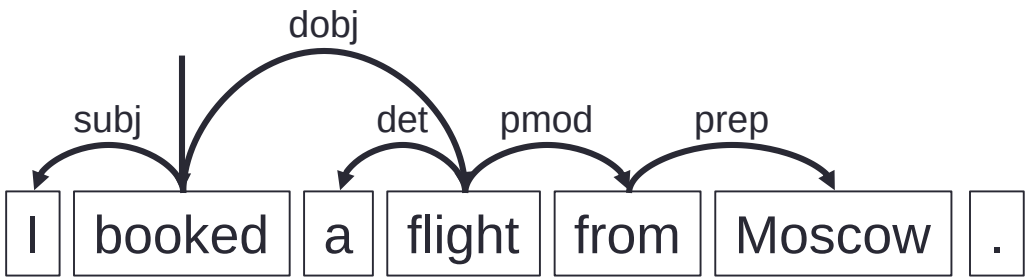
- from
- flight
- booked
- <root>

Буфер

- .

Зависимости

- $(booked, subj, I)$
- $(root, root, booked)$
- $(flight, det, a)$
- $(booked, dobj, flight)$
- $(flight, pmod, from)$
- $(from, prep, Moscow)$



Reduce

Arc-eager

Стек

flight

booked

<root>

Буфер

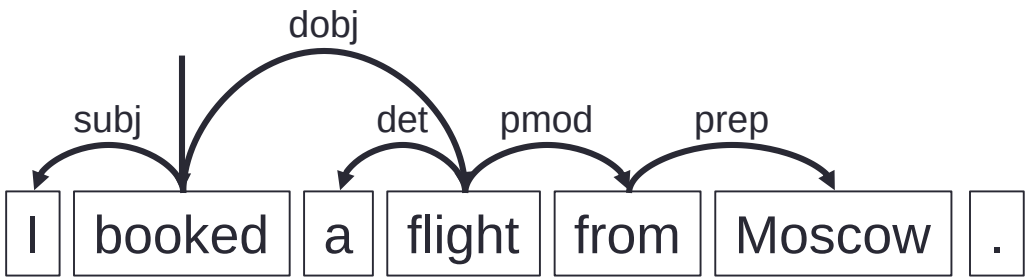
.

Зависимости

$(booked, subj, I)$ $(root, root, booked)$

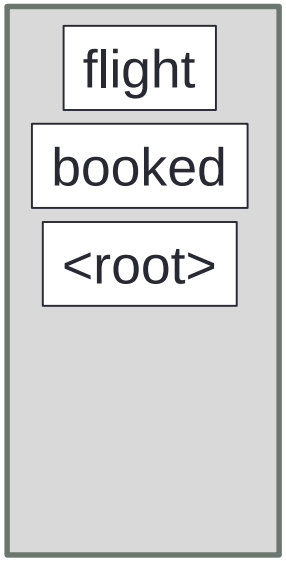
$(flight, det, a)$ $(booked, dobj, flight)$

$(flight, pmod, from)$ $(from, prep, Moscow)$



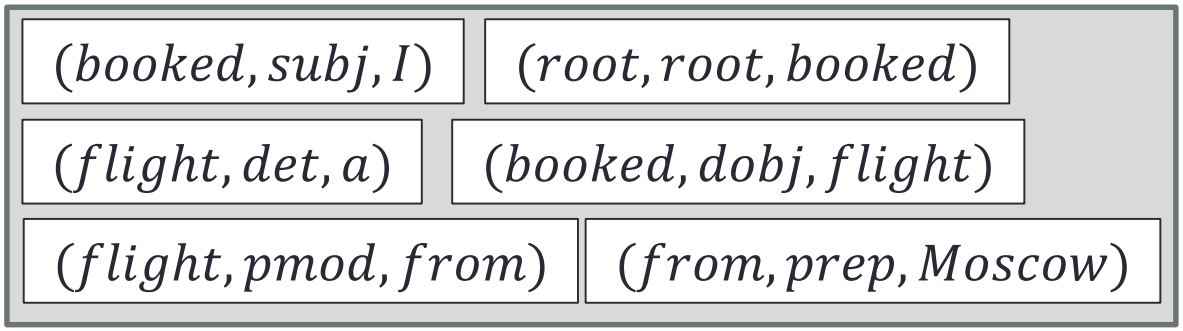
Arc-eager

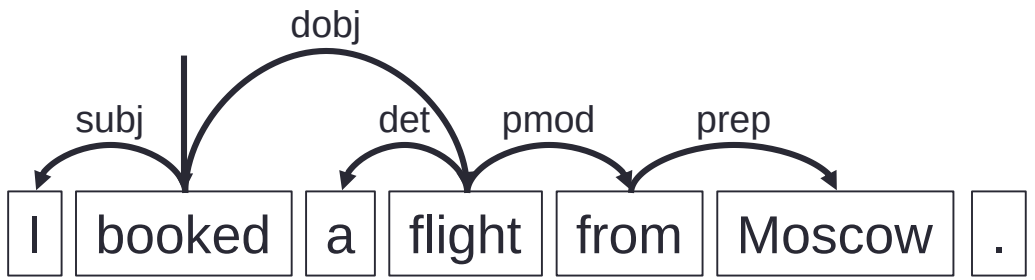
Стек



Буфер

Зависимости





Arc-eager

Стек

booked

<root>

Буфер

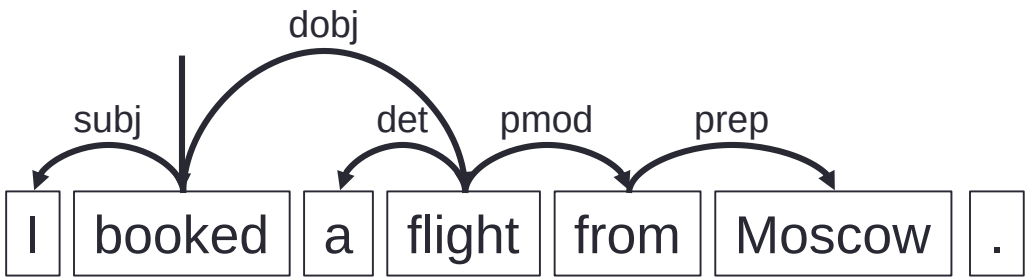
.

Зависимости

$(booked, subj, I)$ $(root, root, booked)$

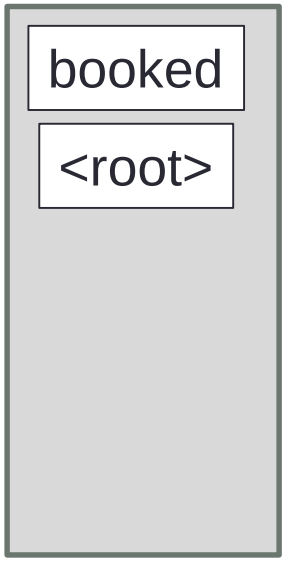
$(flight, det, a)$ $(booked, dobj, flight)$

$(flight, pmod, from)$ $(from, prep, Moscow)$



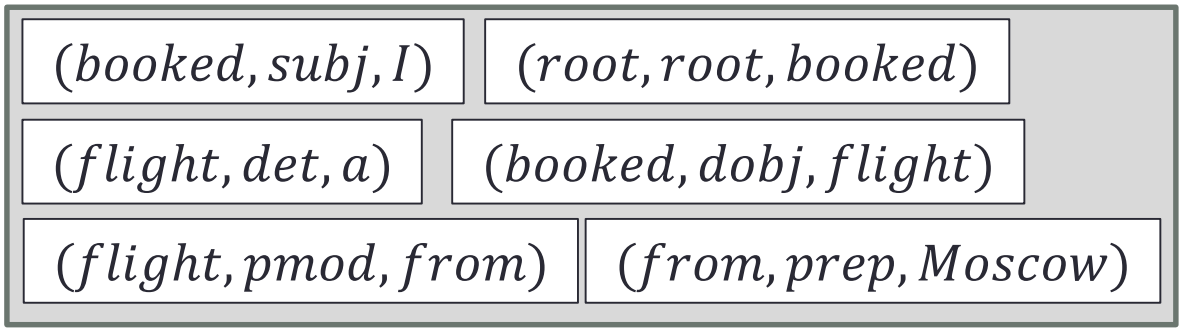
Arc-eager

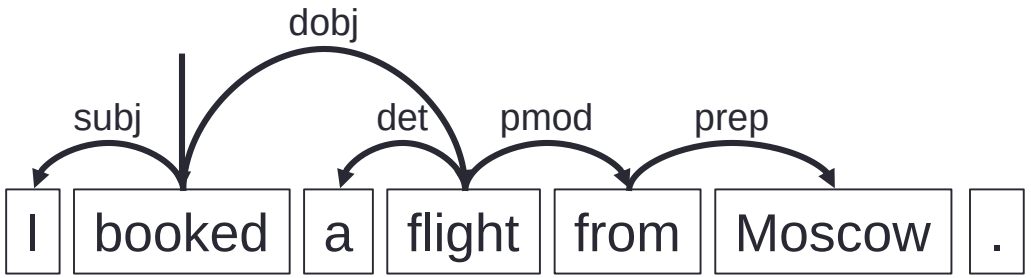
Стек



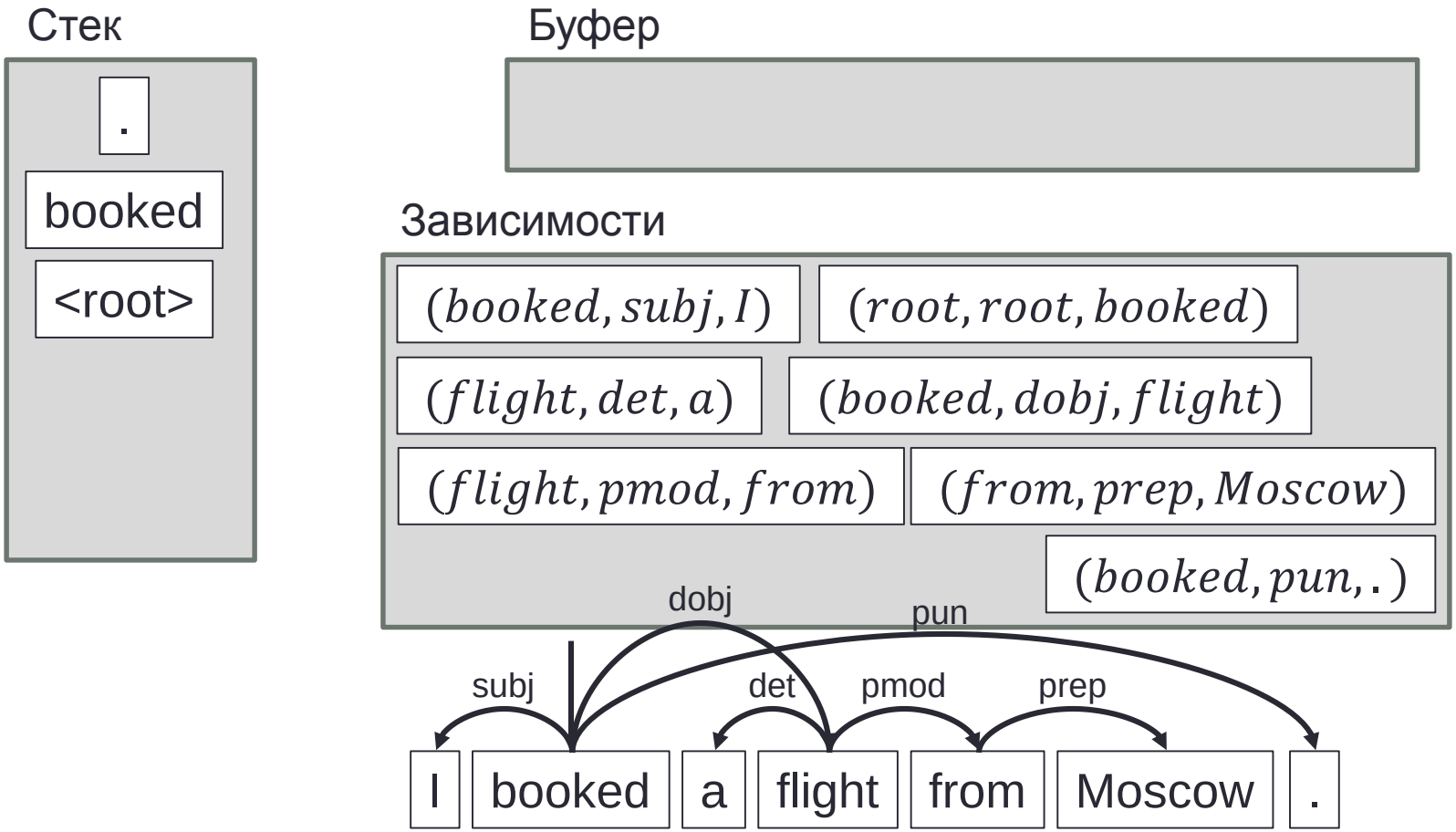
Буфер

Зависимости



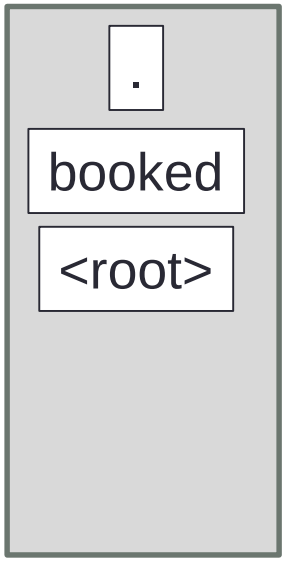


Arc-eager


Стек
.
booked
<root>
Буфер
Зависимости
$(booked, subj, I)$
$(root, root, booked)$
$(flight, det, a)$
$(booked, dobj, flight)$
$(flight, pmod, from)$
$(from, prep, Moscow)$
$(booked, pun, .)$
dobj
pun
subj
det
pmod
prep
I
booked
a
flight
from
Moscow
.

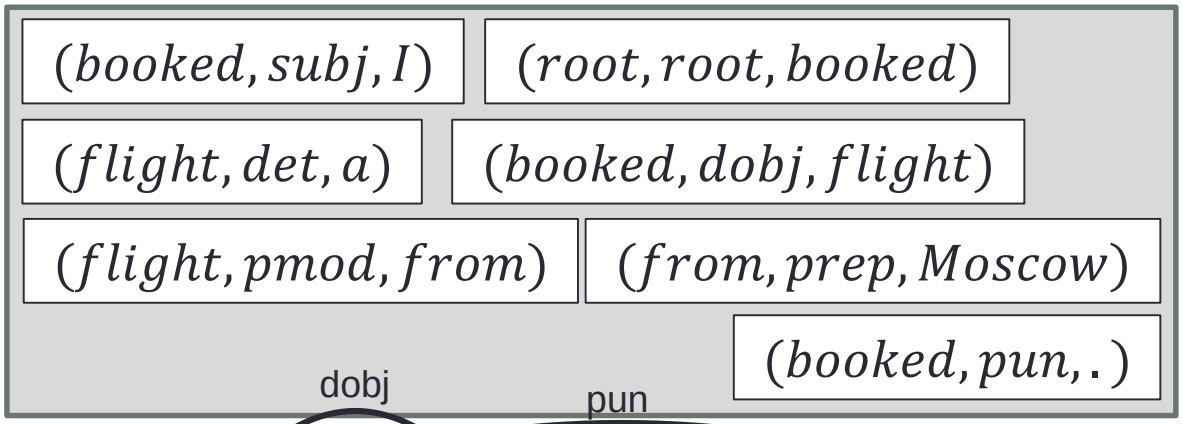
Arc-eager

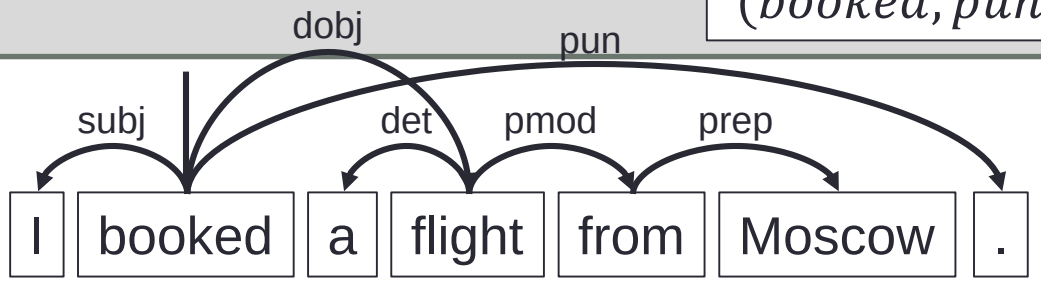
Стек



Буфер

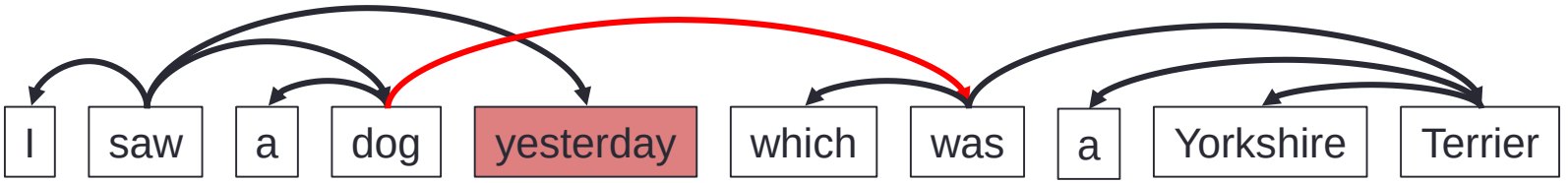
Зависимости





Проективное синтаксическое дерево

- Дерево зависимостей является проективным, если
  - Для каждой дуги в дереве существует направленный путь от главного слова до каждого из слов, расположенных между главным и зависимым словом этой дуги
  - $(i, l, j) \Rightarrow \forall k \in [\min(i, j), \max(i, j)], i \rightarrow^* k$



Non-projective Transition-based Parsing. Attardi's system

- Инициализация
  - $c_s(x = x_1 \dots x_n) = ([0], [1, \dots, n], \emptyset)$
- Конечное состояние
  - $C_t = \{c \in C | c = ([0], [\,], A)\}$
- Переходы (для левых зависимостей требуется $i \neq 0$)
  - $(Shift)$ $(\sigma, [i|\beta], A) \rightarrow ([\sigma|i], \beta, A)$
  - $(LeftArc_l)$ $([\sigma|i|j], B, A) \rightarrow ([\sigma|j], B, A \cup \{(j, l, i)\})$
  - $(RightArc_l)$ $([\sigma|i|j], B, A) \rightarrow ([\sigma|i], B, A \cup \{(i, l, j)\})$
  - $(LeftArc2_l)$ $([\sigma|i|k|j], B, A) \rightarrow ([\sigma|k|j], B, A \cup \{(j, l, i)\})$
  - $(RightArc2_l)$ $([\sigma|i|k|j], B, A) \rightarrow ([\sigma|i|k], B, A \cup \{(i, l, j)\})$
  - $(LeftArc3_l)$ $([\sigma|i|k_1|k_2|j], B, A) \rightarrow ([\sigma|k_1|k_2|j], B, A \cup \{(j, l, i)\})$
  - $(RightArc3_l)$ $([\sigma|i|k_1|k_2|j], B, A) \rightarrow ([\sigma|i|k_1|k_2], B, A \cup \{(i, l, j)\})$

Non-projective Transition-based Parsing. Online reordering

- Инициализация
  - $c_s(x = x_1 \dots x_n) = ([0], [1, \dots, n], \emptyset)$
- Конечное состояние
  - $C_t = \{c \in C | c = ([0], [\,], A)\}$
- Переходы
  - $(Shift)$ $(\sigma, [i|\beta], A) \rightarrow ([\sigma|i], \beta, A)$
  - $(LeftArc_l)$ $([\sigma|i|j], B, A) \rightarrow ([\sigma|j], B, A \cup \{(j, l, i)\})$, если $i \neq 0$
  - $(RightArc_l)$ $([\sigma|i|j], B, A) \rightarrow ([\sigma|i], B, A \cup \{(i, l, j)\})$
  - $(Swap)$ $([\sigma|i|j], \beta, A) \rightarrow ([\sigma|j], [i|\beta], A)$, если $i \neq 0$ и $i < j$

Transition-based Parsing. Предсказание переходов

- Задача многоклассовой классификации
  - Объекты классификации: конфигурации парсера
  - Классы: правила перехода

- Методы:
  - Multiclass SVM
  - Naïve Bayes
  - Decision trees
  - Нейронные сети
  - …

Признаки классификации

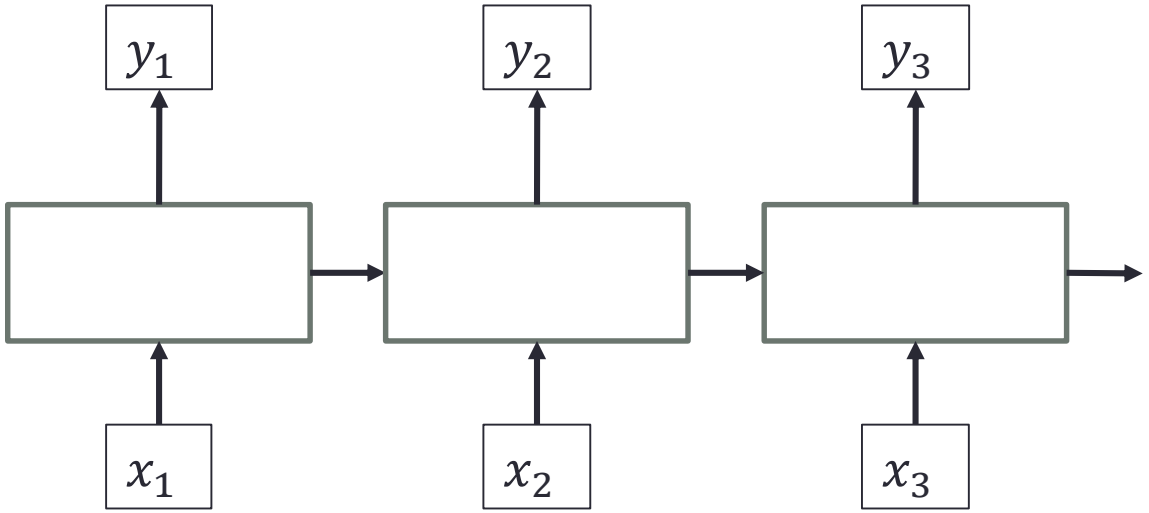
- Признаки для элемента
  - Словоформа
  - Лемма
  - Часть речи
  - Значения грамматических категорий
  - Тип отношения
- Элементы:
  - i-й элемент стека
  - i-й элемент буфера
  - Левое отношение элемента стека
  - Правое отношение элемента стека

Long Short-Term Memory

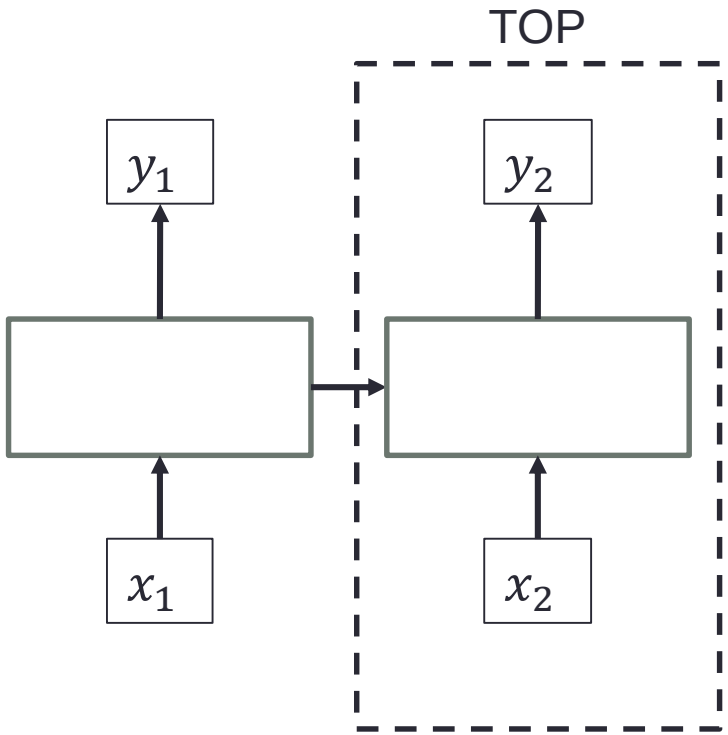
- $i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i)$
- $f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f\right)$
- $c_t = f_t \odot c_{t-1} i_t + \tanh \odot (W_{cx}x_t + W_{ch}h_{t-1} + b_c)$

- $o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_0)$
- $h_t = o_t \odot \tanh(c_t)$

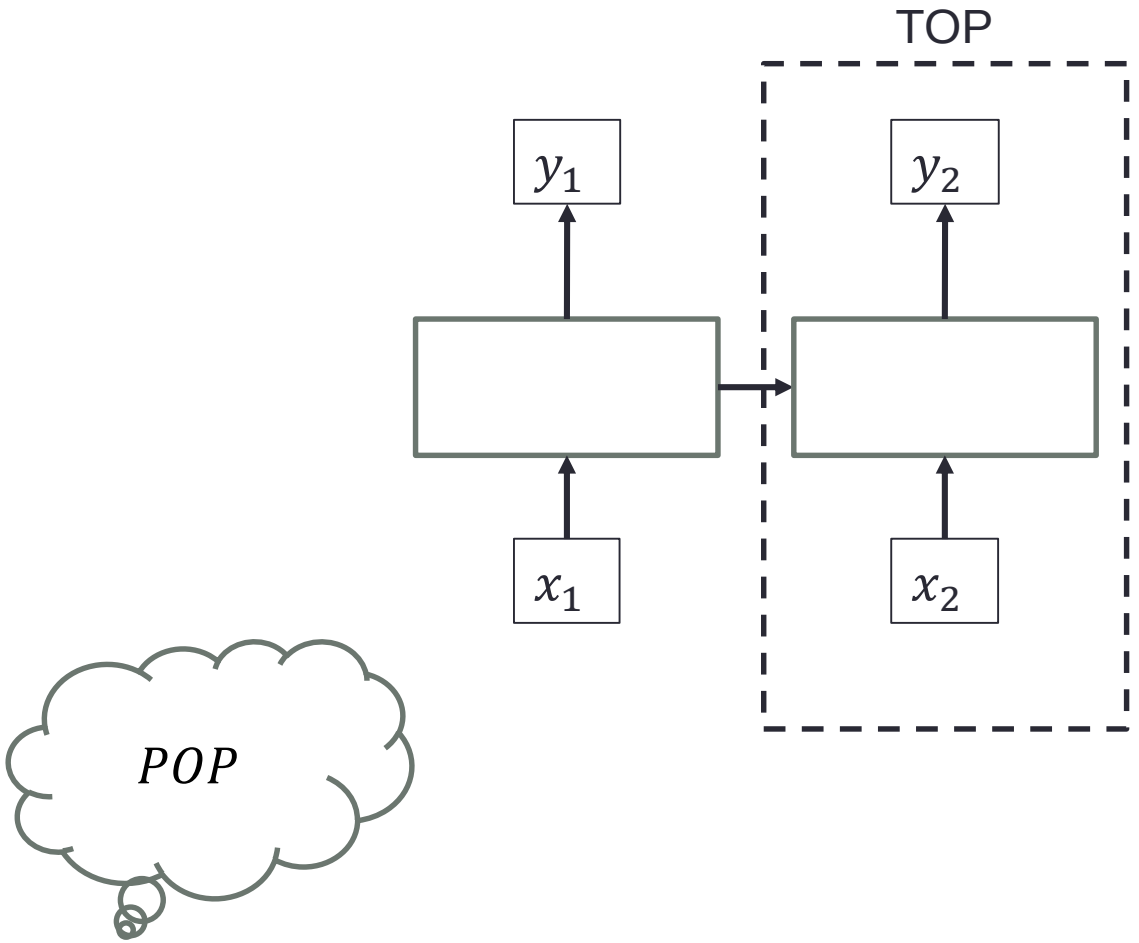
- $y_t = g(h_t)$



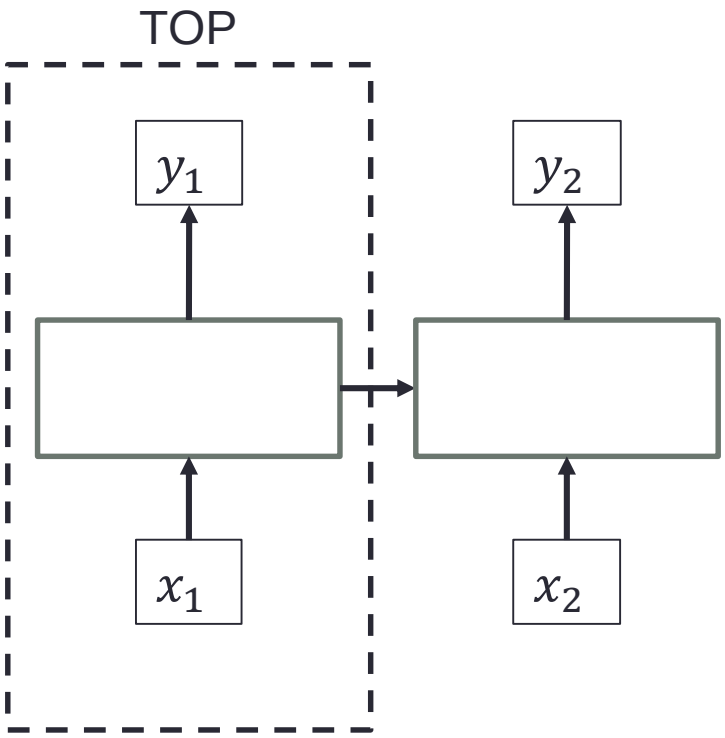
Stack Long Short-Term Memory



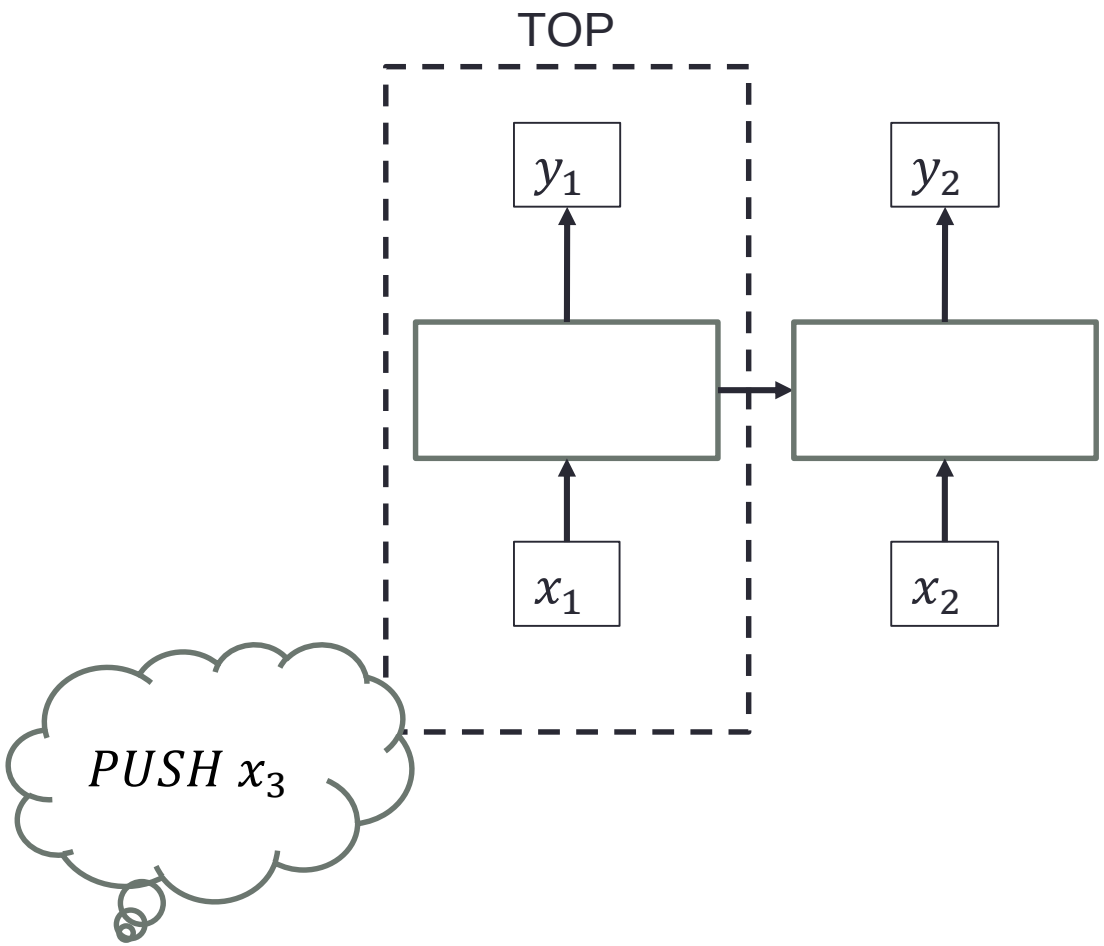
Stack Long Short-Term Memory



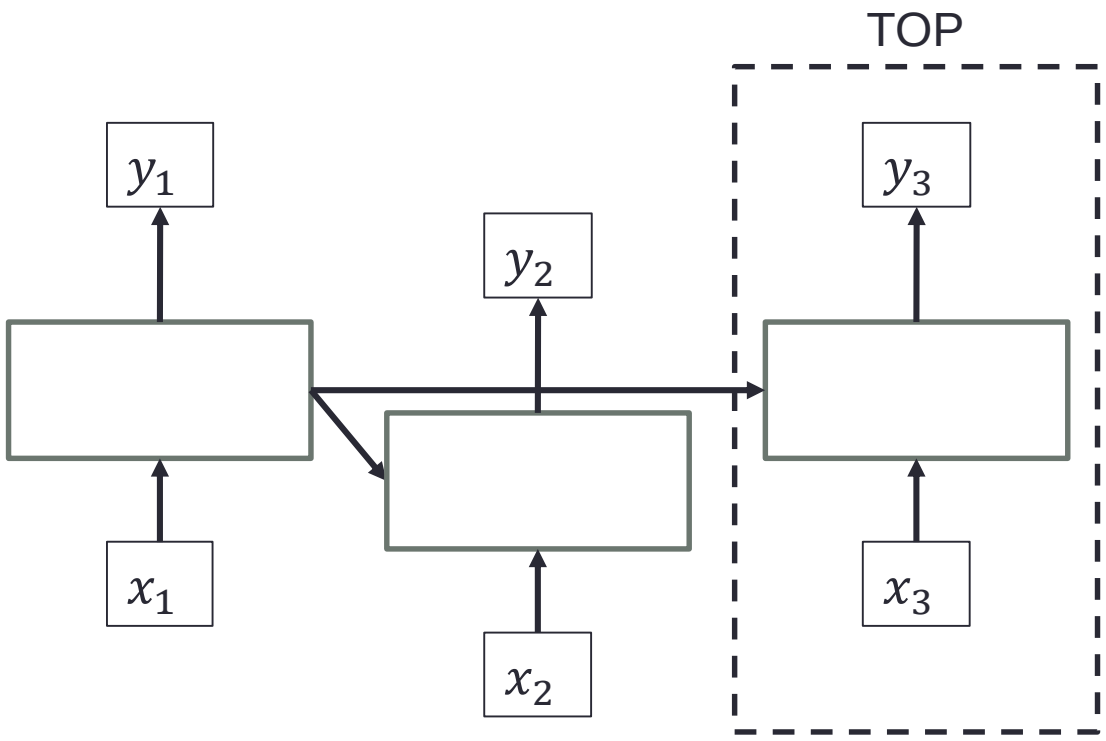
Stack Long Short-Term Memory



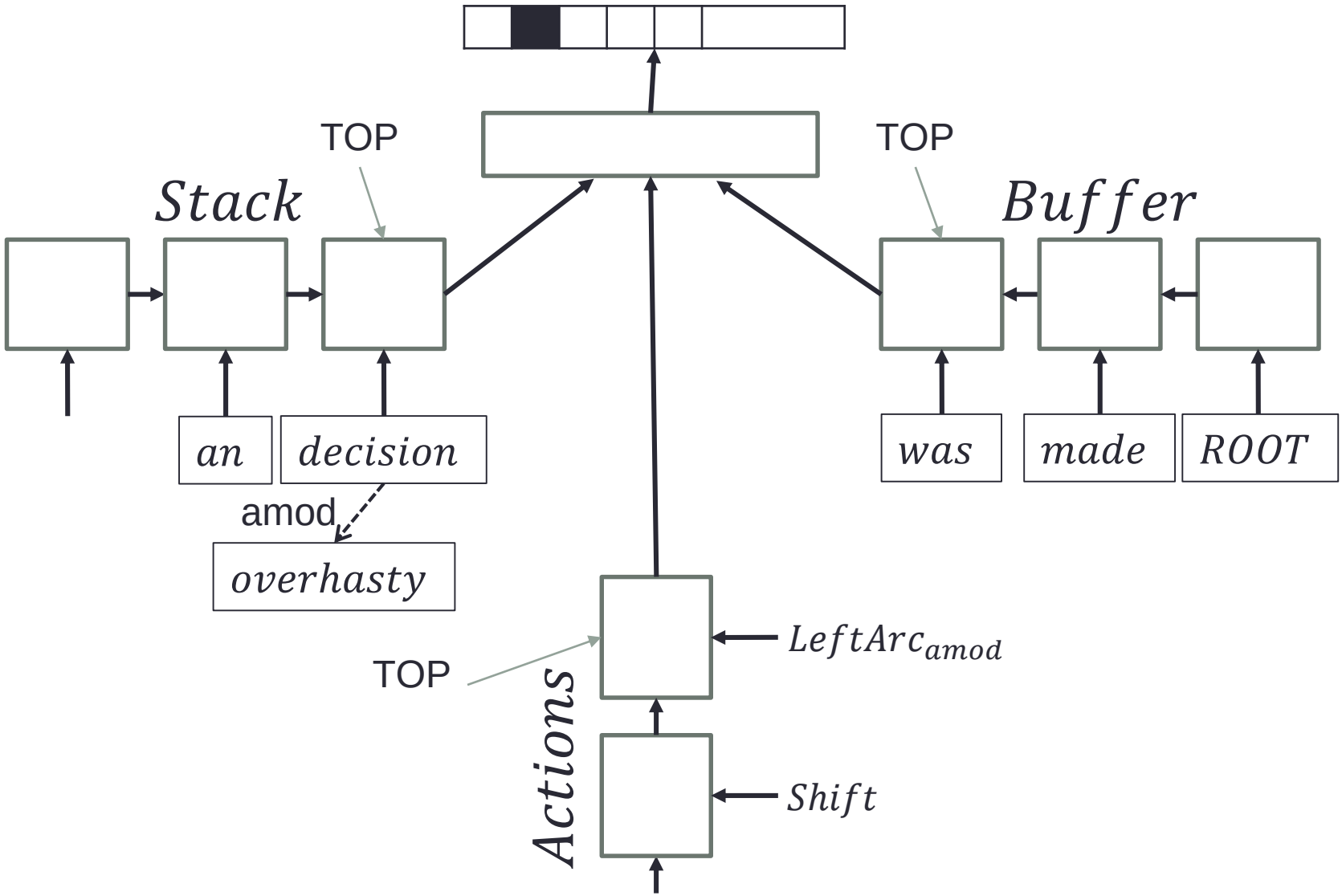
Stack Long Short-Term Memory



Stack Long Short-Term Memory



Stack LSTM Parser



Stack LSTM Parser

- Переходы Arc-standard
  - $Shift$

$$(\sigma, [(v_i, i)|\beta], A) \rightarrow ([\sigma|(v_i, i)], \beta, A)$$

  - $LeftArc_r$

$$\left([\sigma|(v_i, i)|(v_j, j)], B, A\right) \rightarrow \left(\left[\sigma\middle|\left(g_r\left(v_i, v_j\right), j\right)\right], B, A \cup \{j \overset{r}{\rightarrow} i\}\right)$$

  - $RightArc_r$

$$\left([\sigma|(v_i, i)|(v_j, j)], B, A\right) \rightarrow \left(\left[\sigma\middle|\left(g_r\left(v_j, v_i\right), i\right)\right], B, A \cup \{i \overset{r}{\rightarrow} j\}\right)$$

Оценка качества

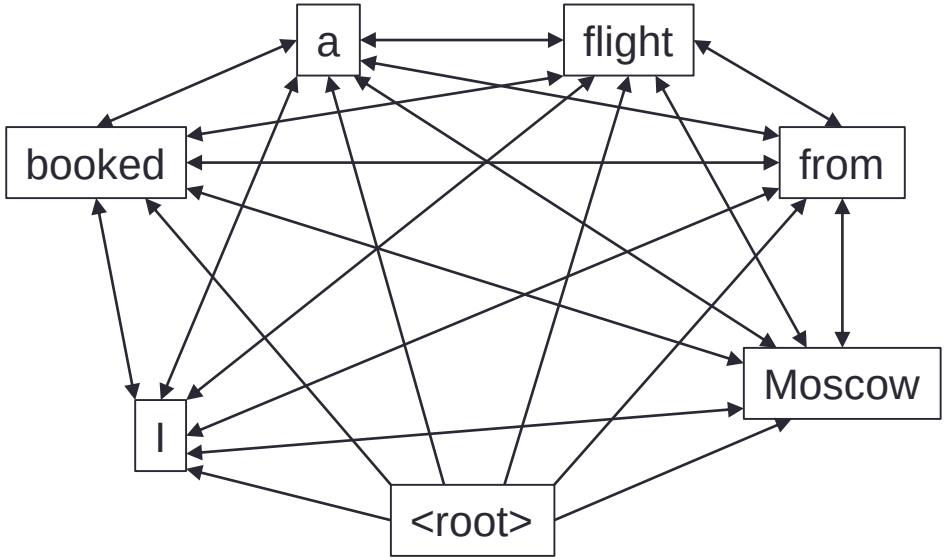
- Labeled attachment score (LAS):
  - Отношение количества правильно построенных дуг к общему количеству дуг
- Labeled exact match (LEM):
  - Отношение количества правильно построенных деревьев к общему количеству деревьев
- Unlabeled attachment score/exact match (UAS/UEM):
  - Те же метрики, но дуги рассматриваются без типов

Оценка качества

- Micro-averaged
  - Считаем отношение количества правильных дуг во всем корпусе к общему количеству дуг в этом корпусе
- Macro-averaged
  - Считаем метрику для каждого предложения и потом усредняем

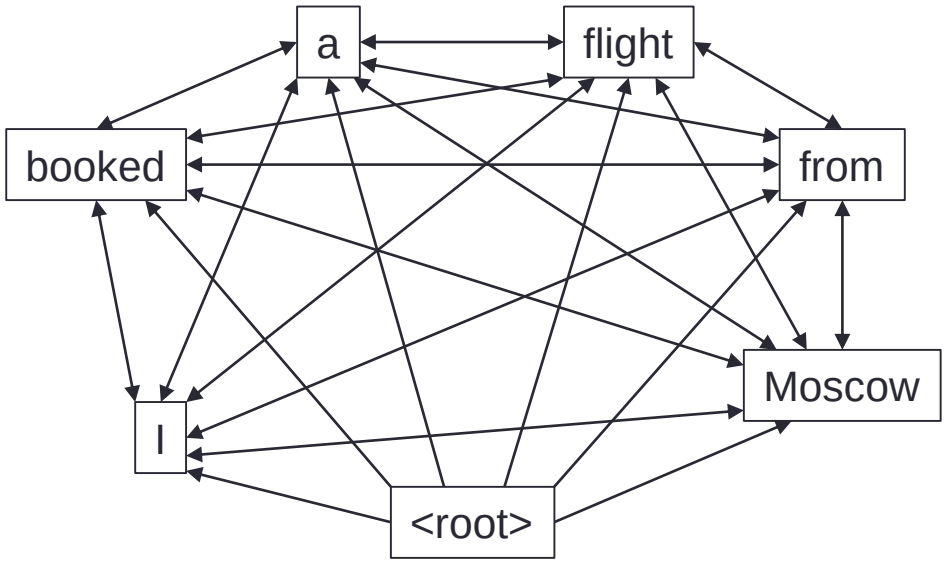
Графовые методы

- Предложение представляется в виде взвешенного полного графа (ориентированного)
  - Вершины графа – слова + <root>
  - Дуги графа – отношения между словами
- Цель
  - Найти подграф, являющийся деревом зависимостей



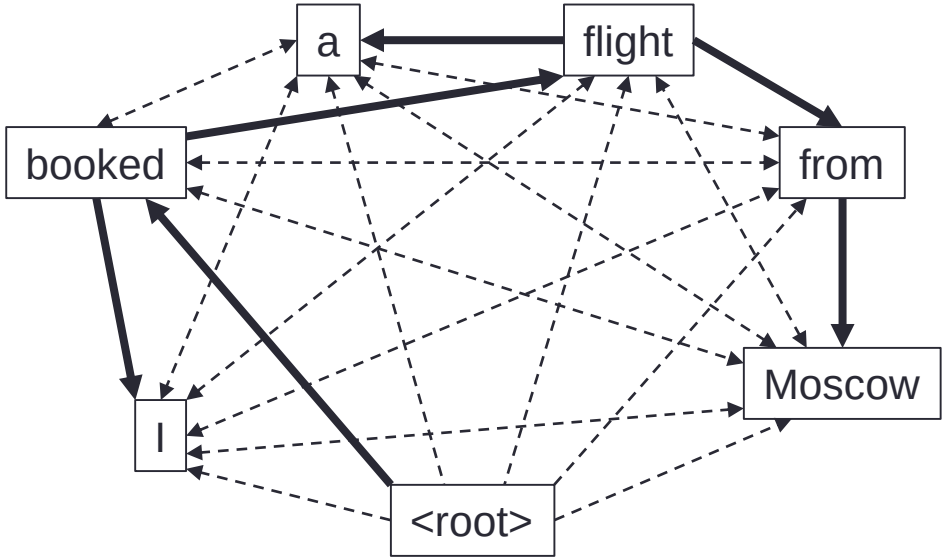
Графовые методы

- Для каждой дуги (i, j) вычисляется ее вес $s(i, j)$
- Вес подграфа в графе $s(x, y) = \sum_{(i,j) \in y} s(i, j)$, где
  - x – вершины подграфа,
  - y – дуги подграфа



Максимальное остовное дерево (MST)

- Задача синтаксического анализа формулируется как задача нахождения максимального остовного дерева (MST) в связанном взвешенном графе
  - Дерево называется остовным, если
    - Включает все вершины графа
    - Содержит минимальное количество дуг из графа, так, чтобы из корня дерева можно было «дойти» до любой вершины графа



Алгоритм Эдмондса

- На входе:
  - Ориентированный граф $D = \langle V, E \rangle$
  - Вершина $r \in V$ – корень
  - Вещественные значения весов $w(e), \forall e \in E$
- На выходе:
  - Остовное ориентированное дерево A минимального веса $w(A) = \sum_{e \in A} w(e)$ с корнем r

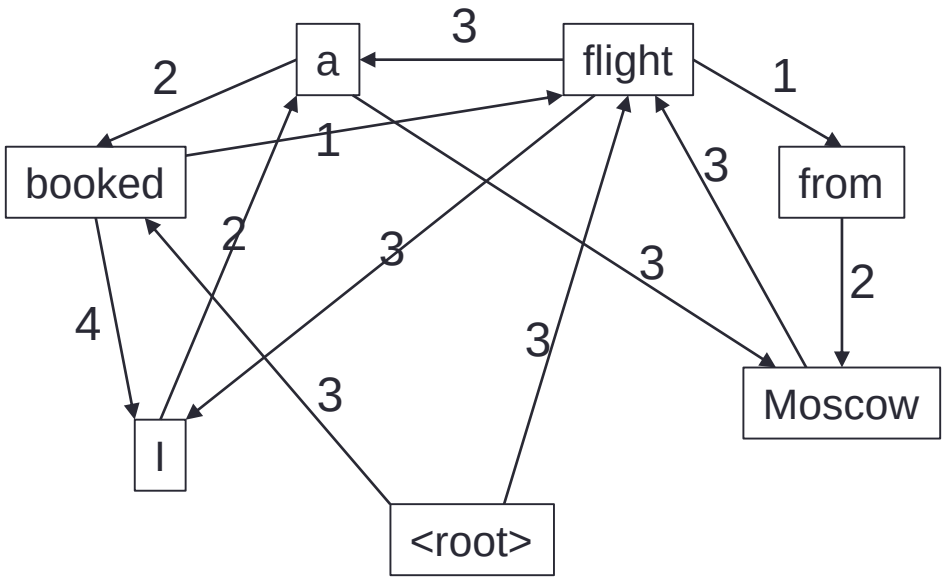
Алгоритм Эдмондса

- Пусть $f(\langle V, E\rangle, r, w)$ возвращает ориентированное дерево минимального веса с корнем в r
  1. Удалить все $(u, r) \in E$
  2. $\pi(v) = \arg\min_{u \in V} w((u, v)) \ \forall v \in V$
  3. $P = \{(\pi(v), v) | v \in V \setminus \{r\}\}$. Если в P нет циклов, то P – искомое дерево
  4. C – произвольный цикл в P. Построим новый граф $D' = \langle V', E'\rangle$. $V' = \{v | v \in V, v \notin C\} \cup \{v_C\}$. E' построим так:
     - $(u, v) \in E, u \notin C, v \in C \Rightarrow$ добавляем в E' ребро $e = (u, v_C)$, $w'(e) = w(u, v) - w(\pi(v), v)$
     - $(u, v) \in E, u \in C, v \notin C \Rightarrow$ добавляем в E' ребро $e = (v_C, v)$, $w'(e) = w(u, v)$
     - $(u, v) \in E, u \notin C, v \notin C \Rightarrow$ добавляем в E' ребро $e = (u, v)$, $w'(e) = w(u, v)$
  5. $A' = f(\langle V', E'\rangle, r, w')$

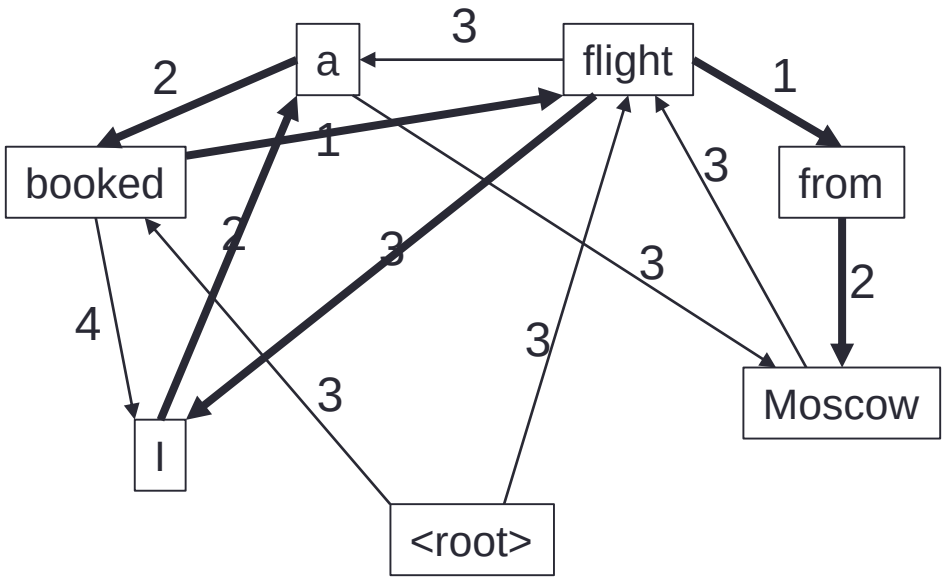
Алгоритм Эдмондса

- A' – ориентированное дерево, поэтому каждая $v \in V', v \neq r$ имеет одно входящее ребро (u, v)
- $(u, v_C) \in A'$ соответствует ребру $(u, v) \in E, v \in C$
- Удалим ребро $(\pi(v), v)$ из C
- Построим P' - искомое дерево:
  - Добавим в P' ребра из C
  - Добавим в P' все ребра, соответствующие ребрам из A'

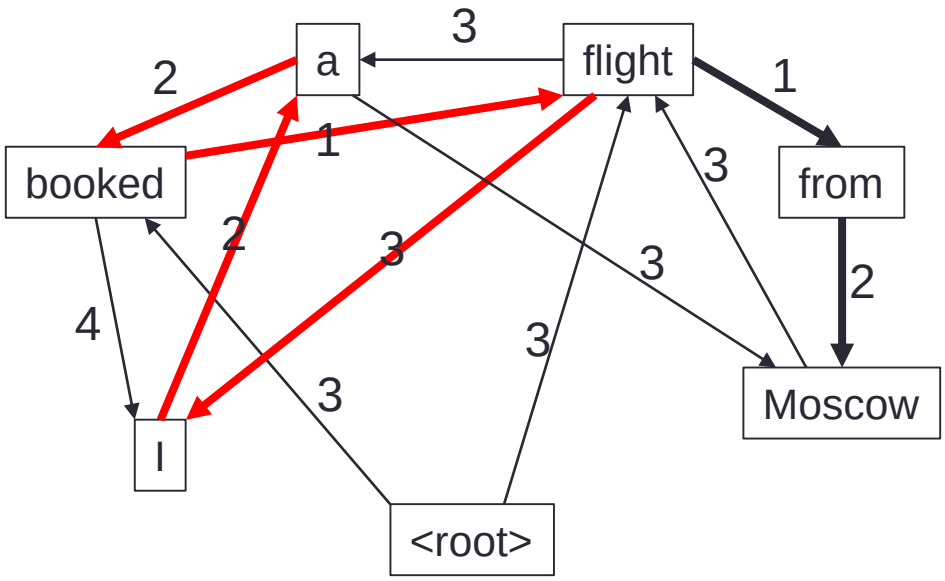
Алгоритм Эдмондса



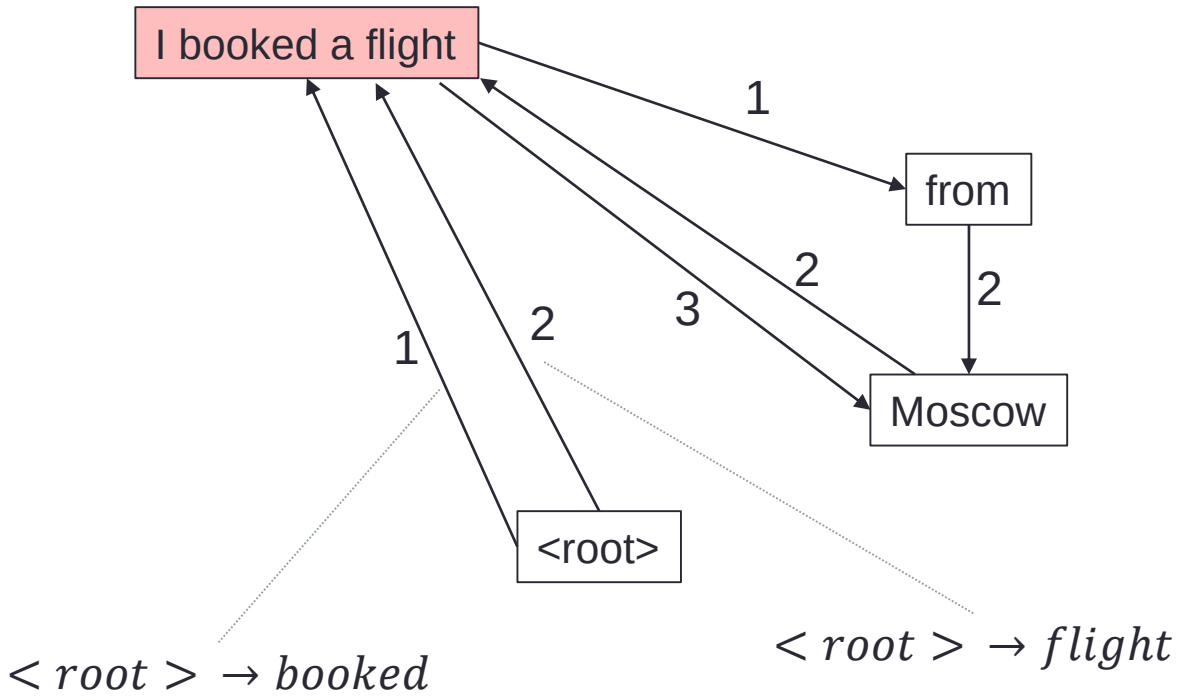
Алгоритм Эдмондса



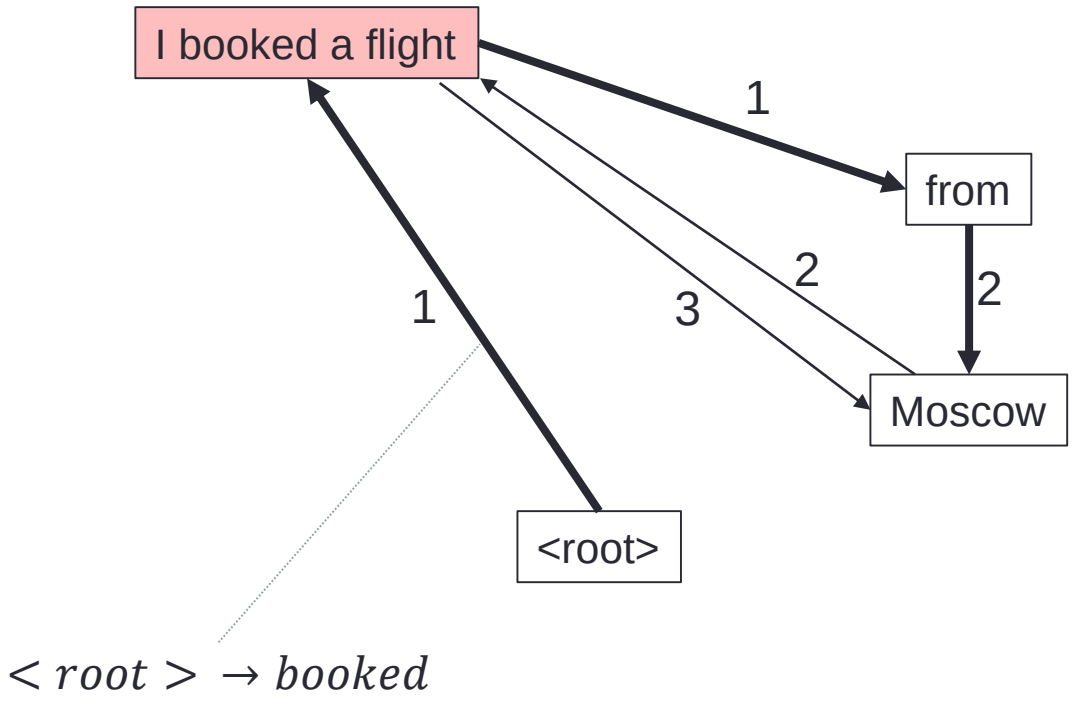
Алгоритм Эдмондса



Алгоритм Эдмондса



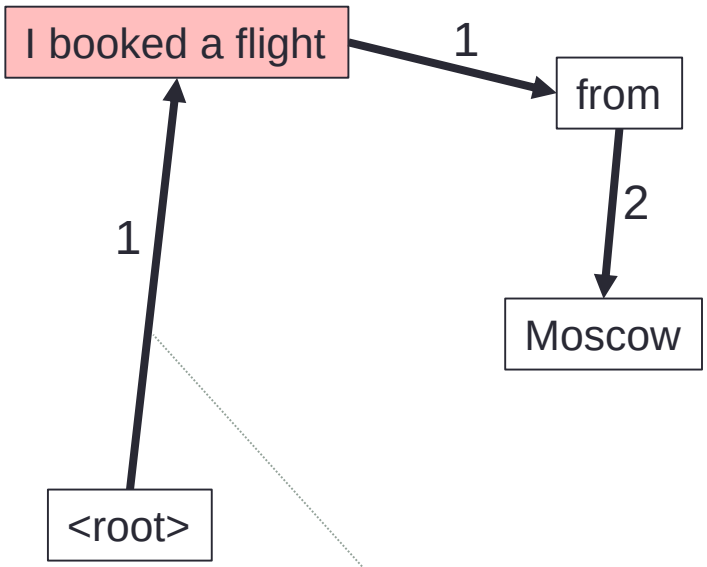
Алгоритм Эдмондса



Алгоритм Эдмондса

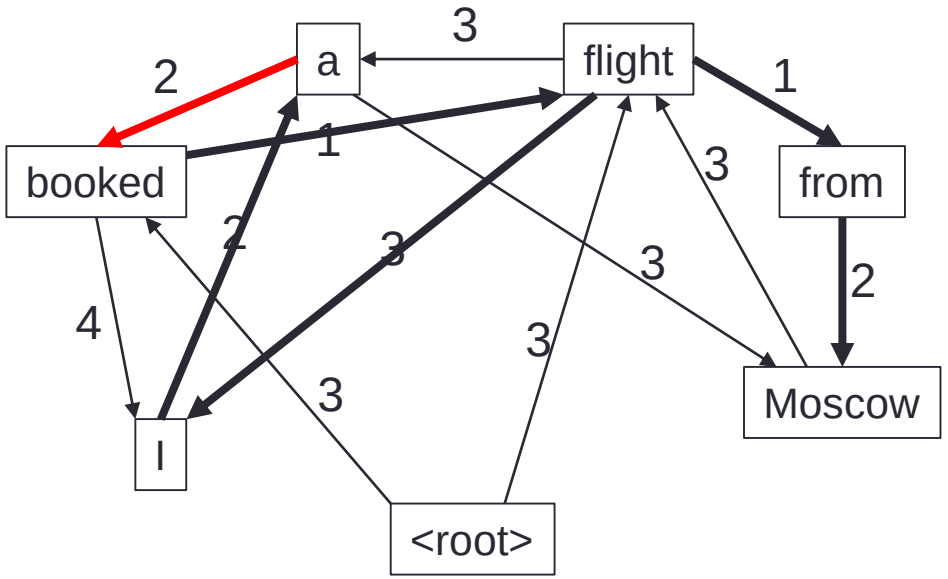
Удалим ребро $(\pi(v), v)$из C

A'

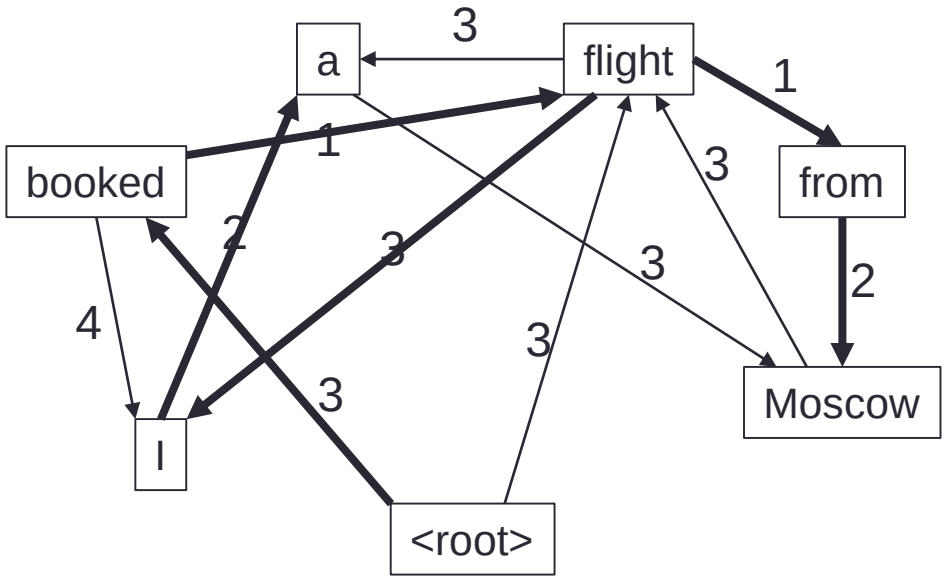


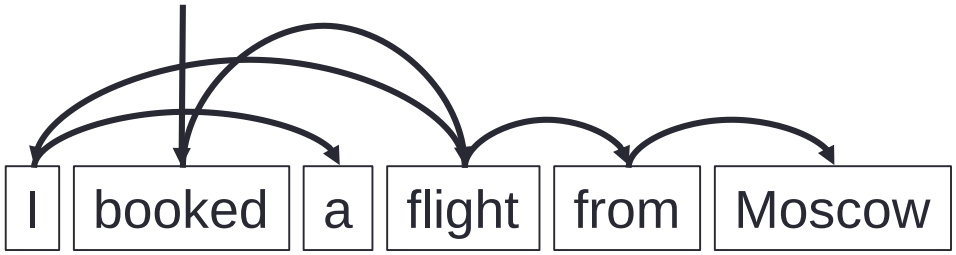
$< root > \rightarrow booked$

D

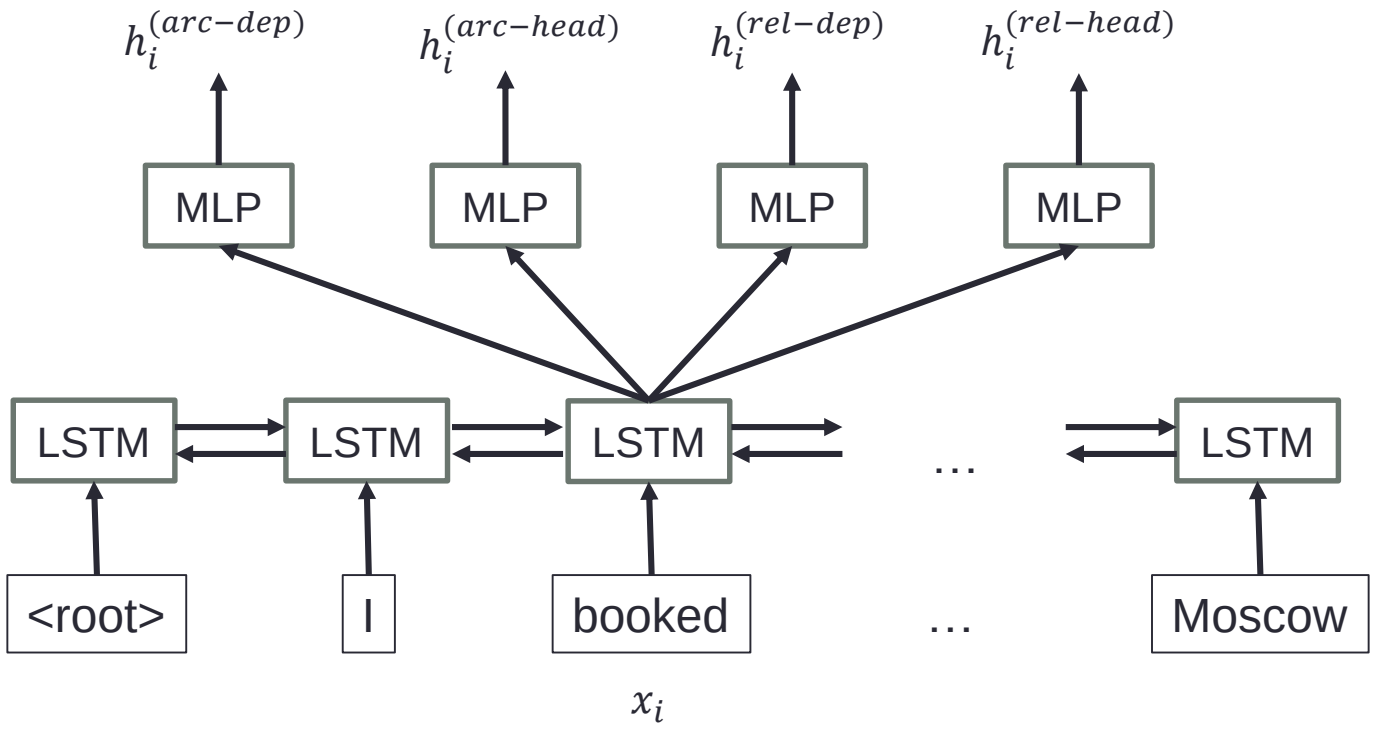


Алгоритм Эдмондса

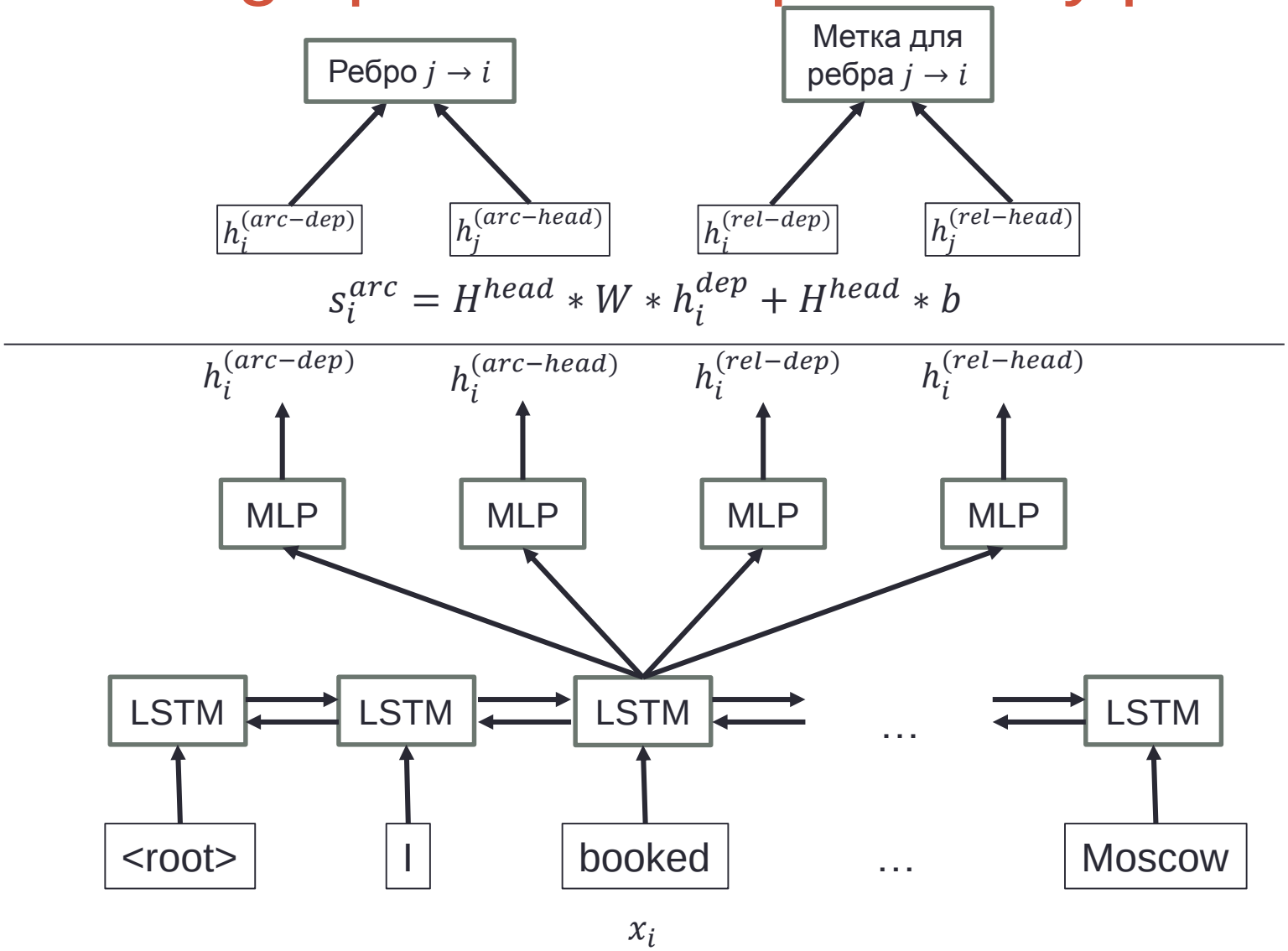




Biaffine graph-based dependency parser



Biaffine graph-based dependency parser



Следующая лекция

Языковые модели на основе нейронных сетей