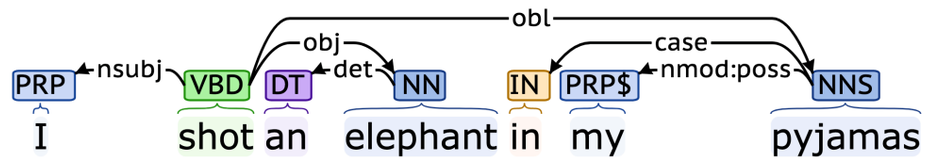
Dependency Grammar

COMP90042

Natural Language Processing

Lecture 16



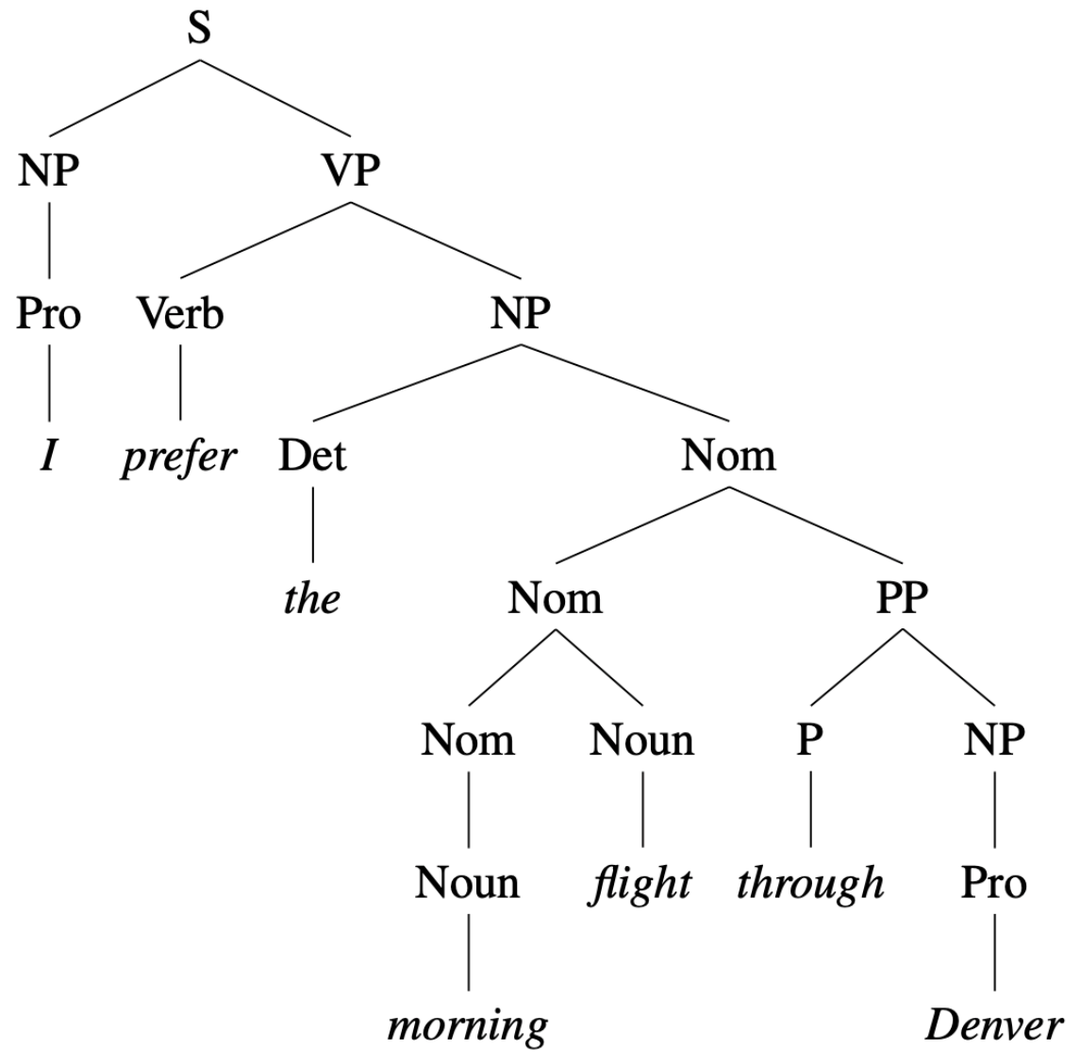
Correction on Lecture 13, Page 8

Regular Languages

- Regular language: the simplest class of languages
- Any **regular expression** is a regular language
- ~~The regular expression itself is the **grammar**~~
  - Evaluates whether a string is part of the language

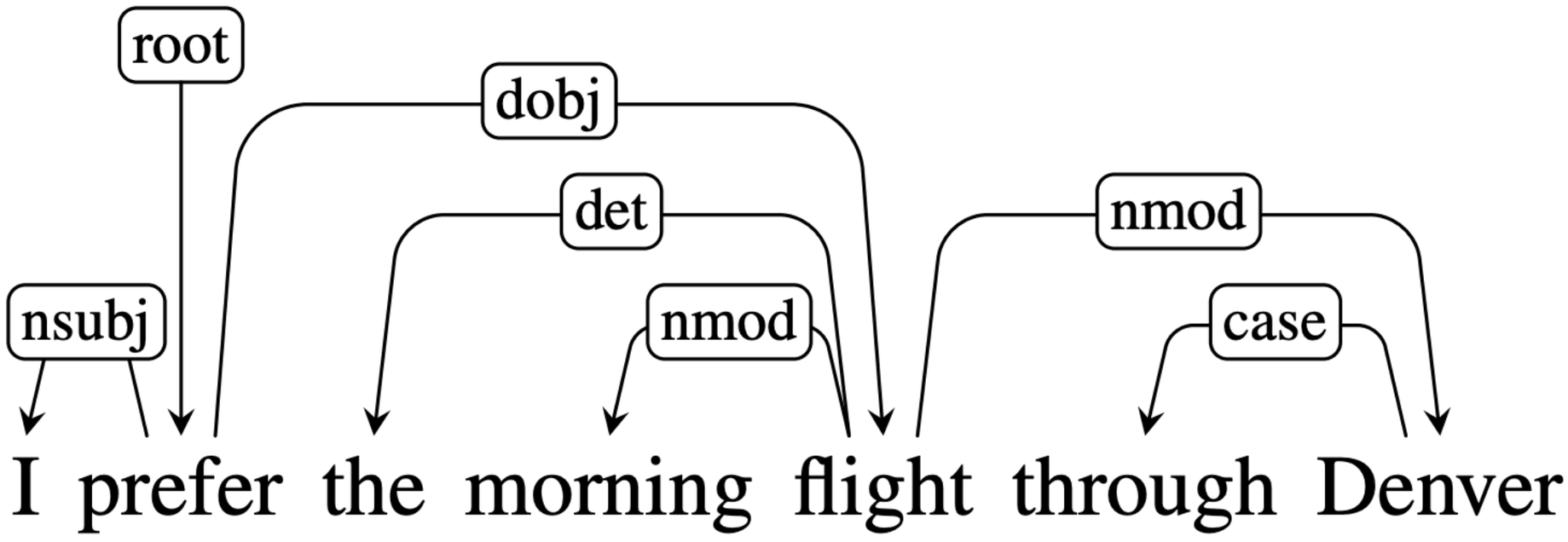
Context-Free Grammars (Recap)

- CFGs assume a constituency tree which identifies the **phrases** in a sentence
  - based on idea that these phrases are interchangeable (e.g., swap an NP for another NP) and maintain grammaticality



Dependency Grammars

- Dependency grammar offers a simpler approach
  - describe **relations** between pairs of words
  - namely, between **heads** and **dependents**
  - e.g. (*prefer*, dobj, *flight*)



Why?

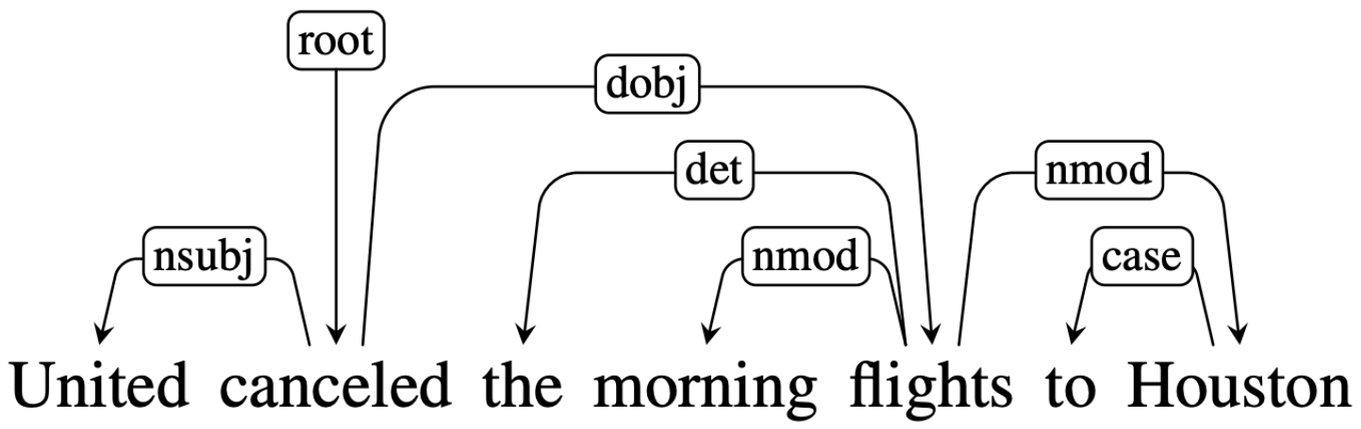
- Deal better with languages that are morphologically rich and have a relatively free word order
  - CFG need a separate rule for each possible place a phrase can occur in
- Head-dependent relations similar to semantic relations between words
  - More useful for applications: coreference resolution, information extraction, etc

Dependency Relations

- Captures the grammatical relation between a head and a dependent
- Head = central word
- Dependent = supporting word
- Grammatical relation = subject, direct object, etc
- Many dependency theories and taxonomies proposed for different languages
- **Universal Dependency**: a framework to create a set of dependency relations that are computationally useful and **cross-lingual**

Universal Dependency

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

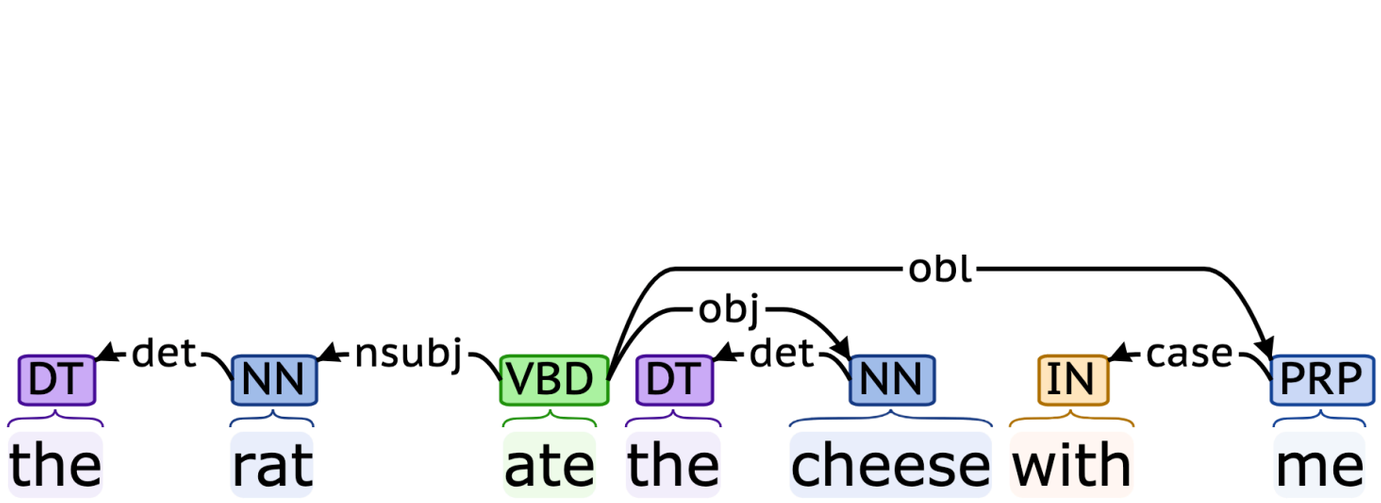


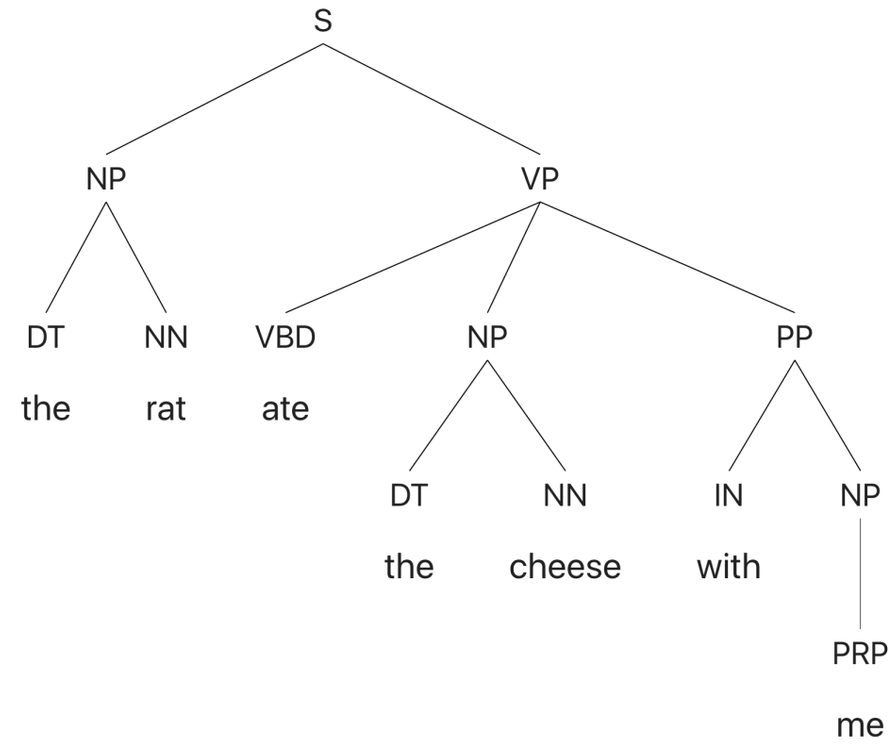
More Examples

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Question Answering

- Dependency tree more directly represents the core of the sentence: *who did what to whom?*
  - captured by the links incident on verb nodes

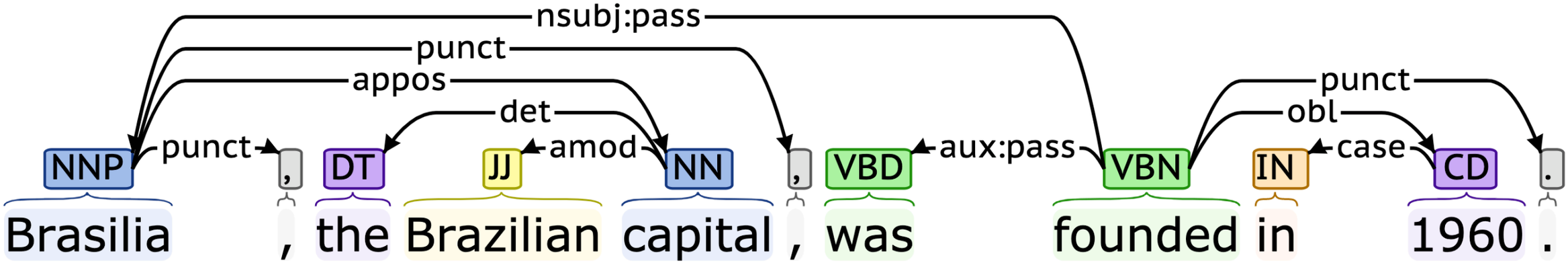




  - more minor details are buried deeper in the tree (e.g., adjectives, determiners etc)

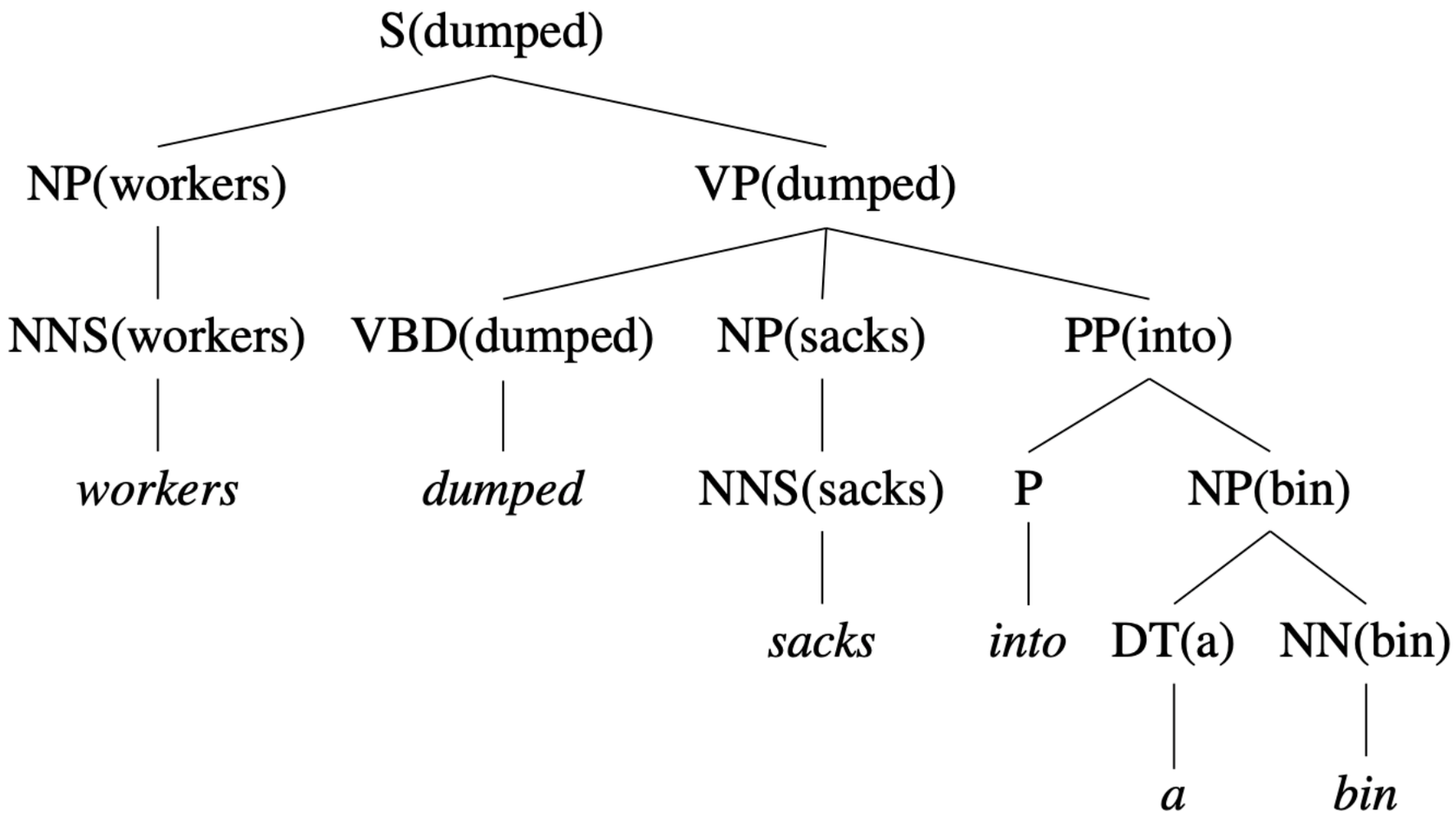
Information Extraction

- “Brasilia, the Brazilian capital, was founded in 1960.”
  - → capital(Brazil, Brasilia)
  - → founded(Brasilia, 1960)
- Dependency tree captures relations succinctly



What about CFGs

- Constituency trees can also provide similar information
- But it requires some distilling using head-finding rules



Dependency vs Constituency

- Dependency tree
  - each node is a word token
  - one node is chosen as the root
  - directed edges link heads and their dependents
- Constituency tree
  - forms a hierarchical tree
  - word tokens are the leaves
  - internal nodes are ‘constituent phrases’ e.g., NP, VP etc
- Both use part-of-speech

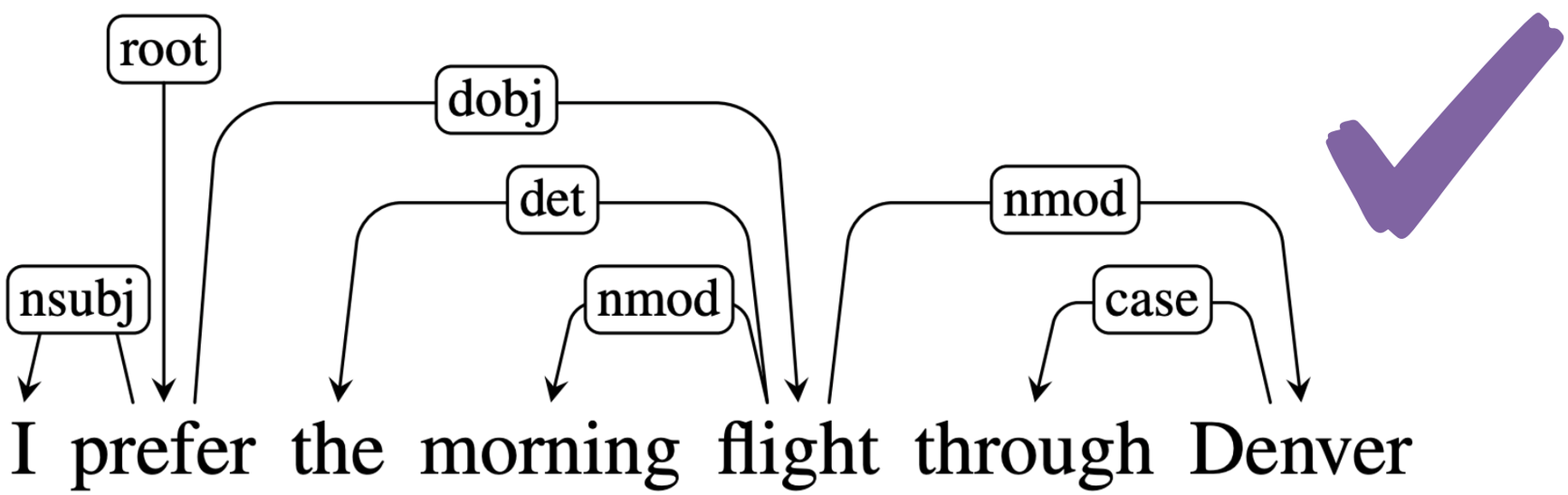
Properties of a Dependency Tree

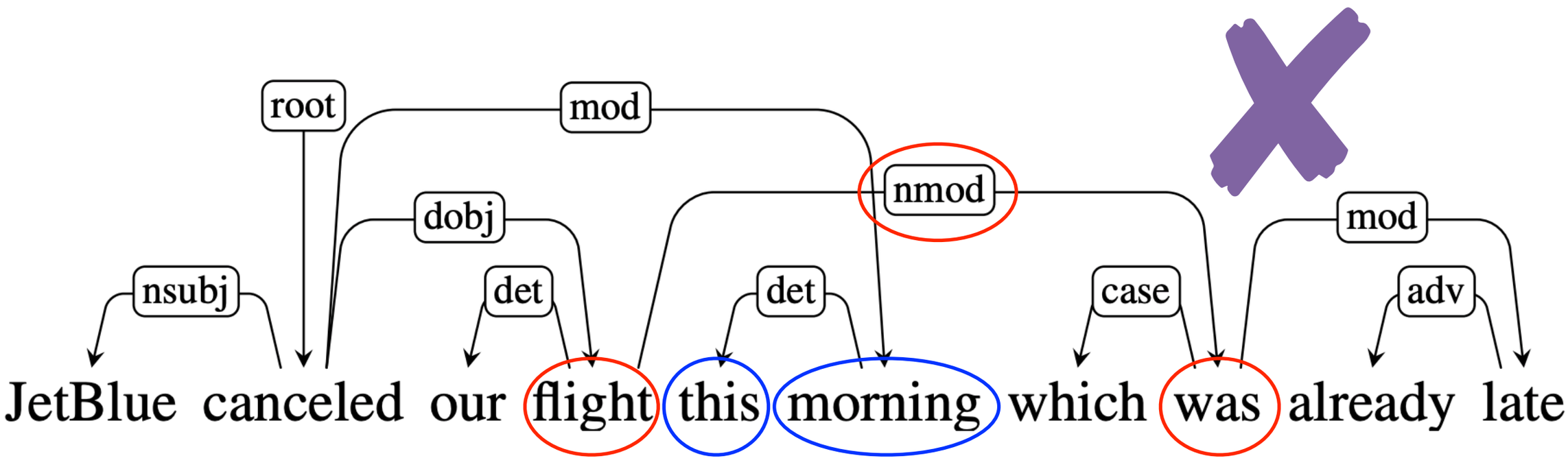
- Each word has a single head (parent)
- There is a single root node
- There is a unique path to each word from the root
- All arcs are **projective**
  - Transition-based parsers can only produce projective trees

Projectivity

- An arc is projective if there is a path from head to every word that lies between the head and the dependent
- A dependency tree is projective if all arcs are projective
- That is, a dependency tree is projective if it can be drawn with no **crossing edges**
- Most sentences are projective, however exceptions exist
  - Common in languages with flexible word order

Projectivity



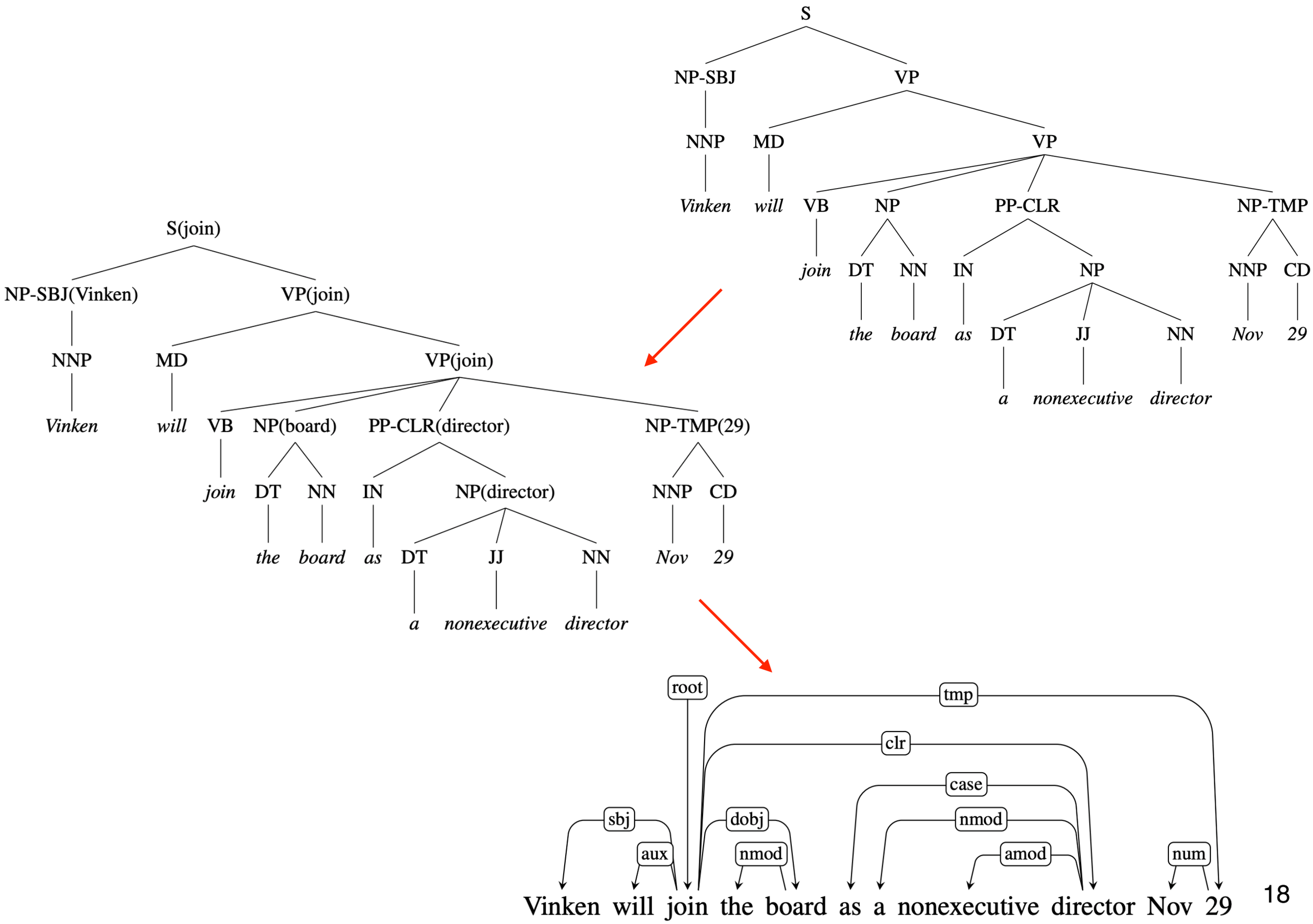


Dependency Treebanks

- A few dependency treebanks
  - Czech, Arabic, Danish, Dutch, Greek, Turkish …
- More recently, **Universal Dependency Treebank**
  - collates >100 treebanks, >60 languages
  - unified part-of-speech, morphology labels, relation types
  - consistent handling of conjunctions and other tricky cases
- http://universaldependencies.org/

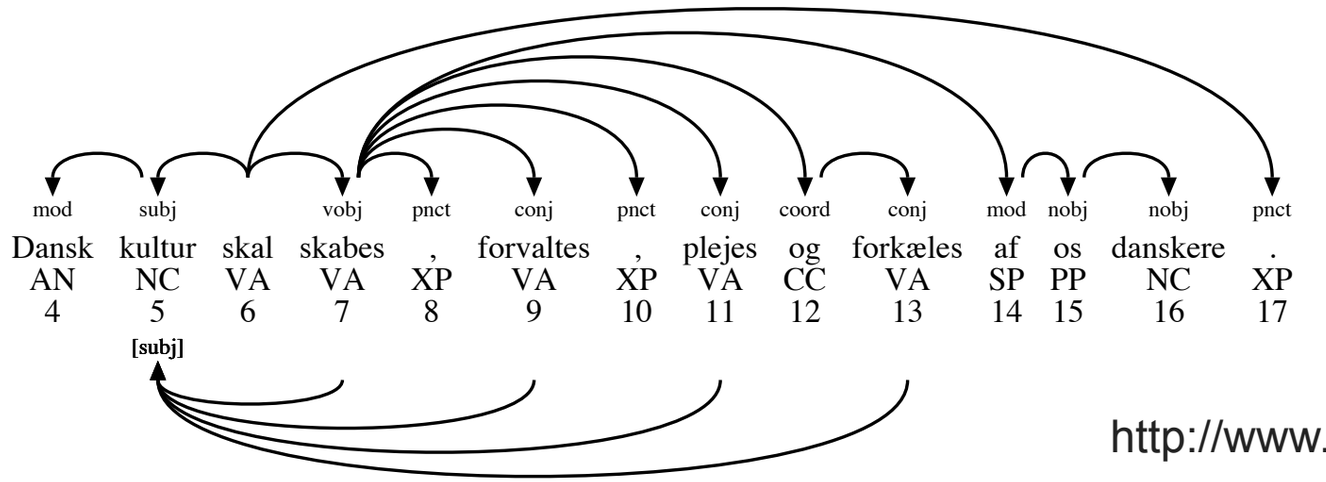
Treebank Conversion

- Many constituency treebanks; some can be automatically converted into dependencies
- Dependency trees generated from constituency trees are always projective
- Main idea: identify head-dependent relations in constituency structure and the corresponding dependency relations
  - Use various heuristics, e.g., head-finding rules
  - often with manual correction


S
NP-SBJ
VP
NNP
MD
VP
Vinken
will
VB
NP
PP-CLR
NP-TMP
join
DT
NN
IN
NP
NNP
CD
the
board
as
DT
JJ
NN
Nov
29
a
nonexecutive
director
S(join)
NP-SBJ(Vinken)
VP(join)
NNP
MD
VP(join)
Vinken
will
VB
NP(board)
PP-CLR(director)
NP-TMP(29)
join
DT
NN
IN
NP(director)
NNP
CD
the
board
as
DT
JJ
NN
Nov
29
a
nonexecutive
director
root
tmp
clr
case
sbj
dobj
nmod
nmod
amod
num
aux
Vinken will join the board as a nonexecutive director Nov 29

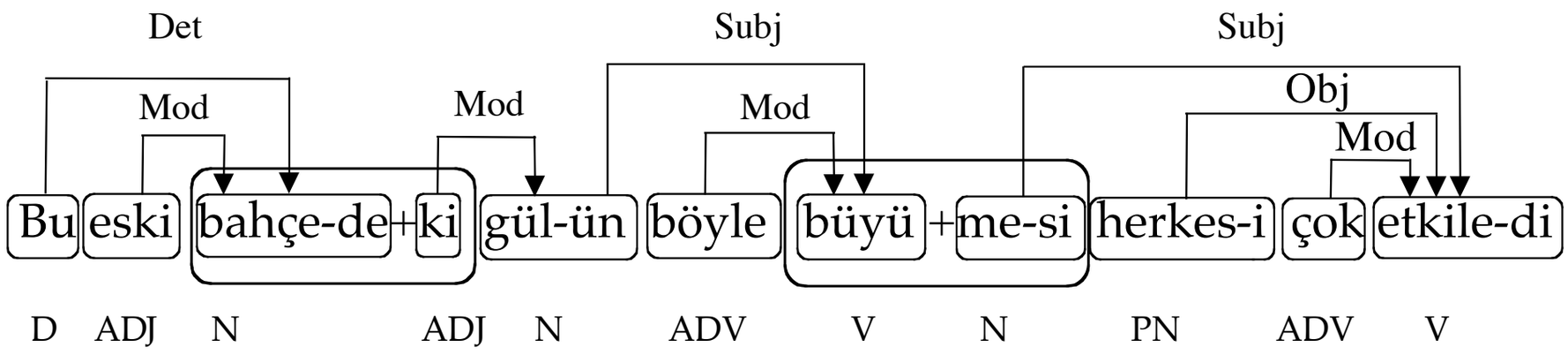
Examples From Treebanks

- Danish DDT includes additional ‘subject’ link for verbs



http://www.buch-kromann.dk/matthias/ddt1.0/

- METU-Sabancı Turkish treebank
  - edges between morphological units, not just words



Oflazer, Kemal, et al. "Building a Turkish treebank." *Treebanks*. Springer, 2003. 261-277.

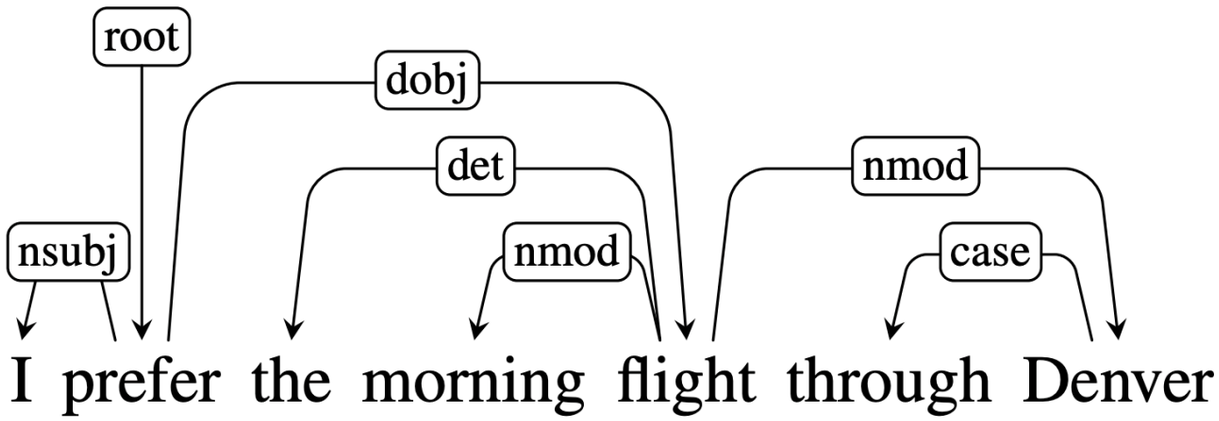
Parsing

Dependency Parsing

- Parsing: task of finding the *best* structure for a given input sentence
  - argmax$_t$ score$(t \mid w)$
- Two main approaches:
  - **transition-based**: treats problem as incremental sequence of decisions over next action in a state machine
  - **graph-based**: encodes problem using nodes/edges and use graph theory methods to find optimal solutions

Transition-Based Parsing: Intuition

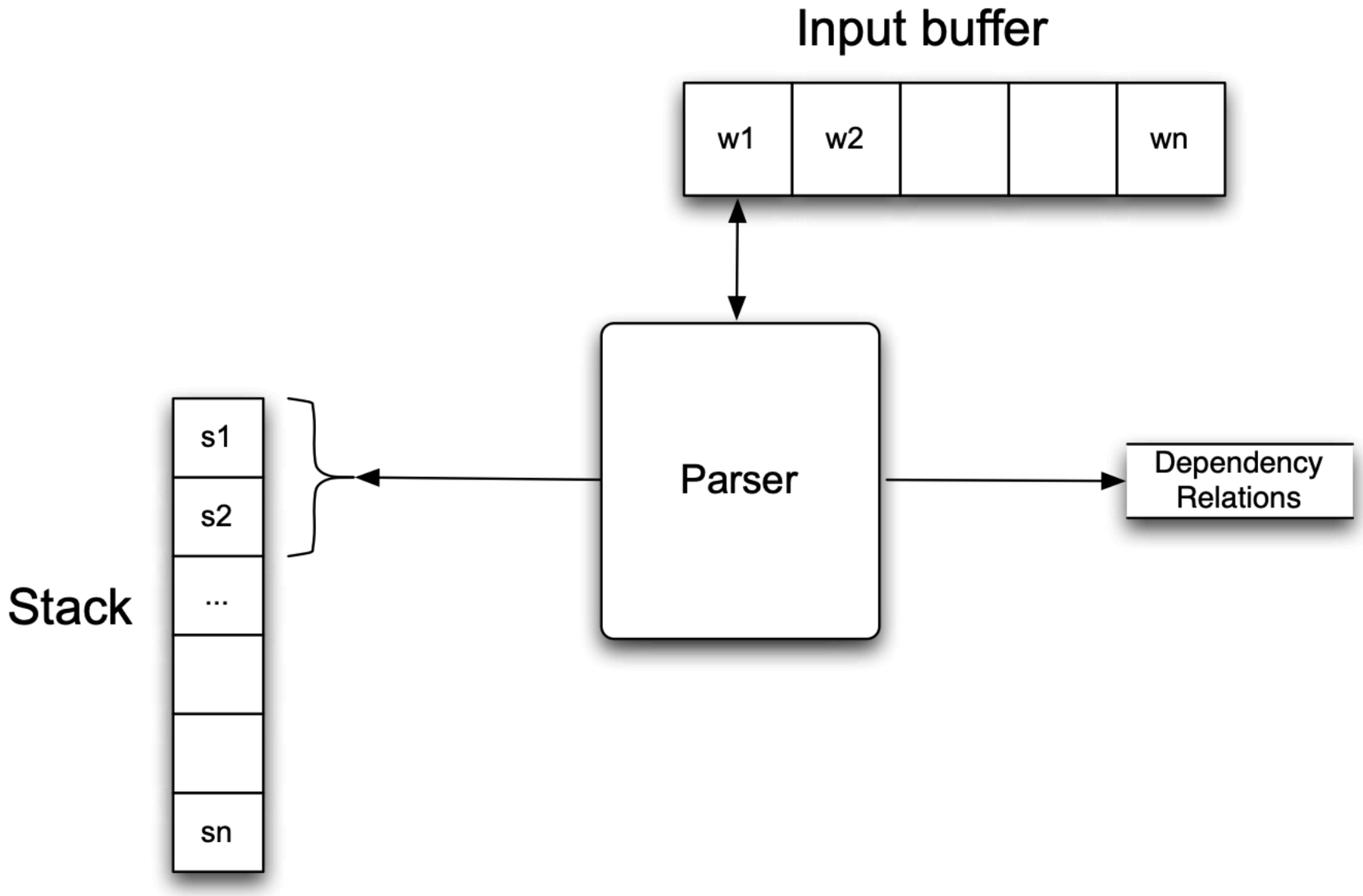
- Examine the words in a single pass from left to right
- At each step, perform one of the following actions:
  - Assign current word as head of some previously seen words **[left arc]**
  - Assign some previously seen words as head of current word **[right arc]**
  - Don’t do anything, store it and move to next step **[shift]**



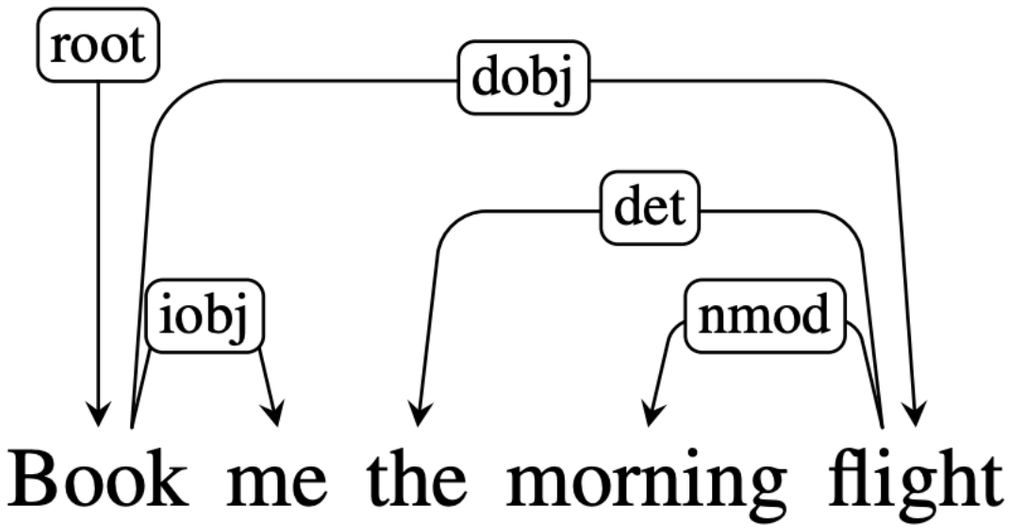
Transition-Based Parsing

- Transition-based parsers can only produce projective trees
- Frames parsing as sequence of transitions
  - maintain two data structures
    - *buffer:* input words yet to be processed
    - *stack:* head words currently being processed
  - two types of transitions
    - *shift:* move word from buffer to top of stack
    - *arc:* add arc (left/right) between top two items on stack, and *remove* dependent from stack

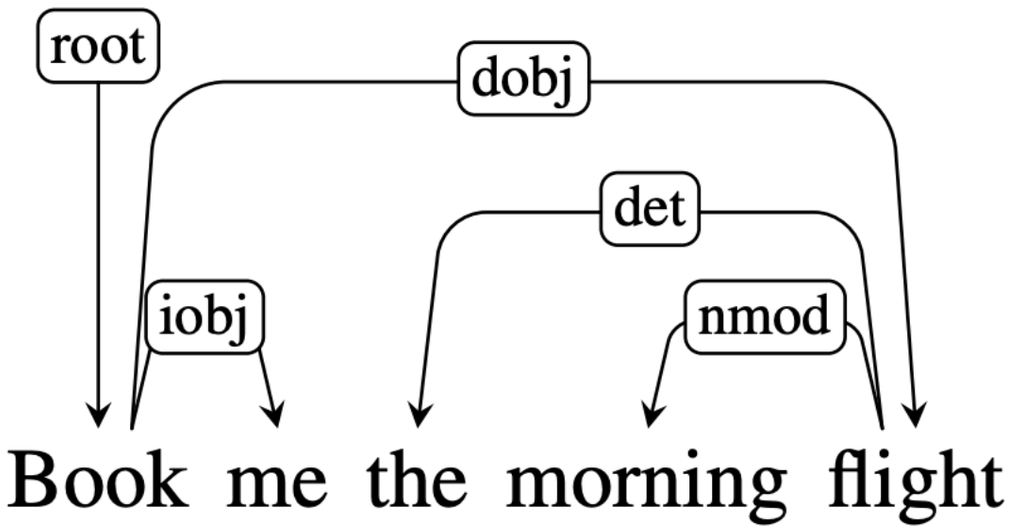
Transition-Based Parsing



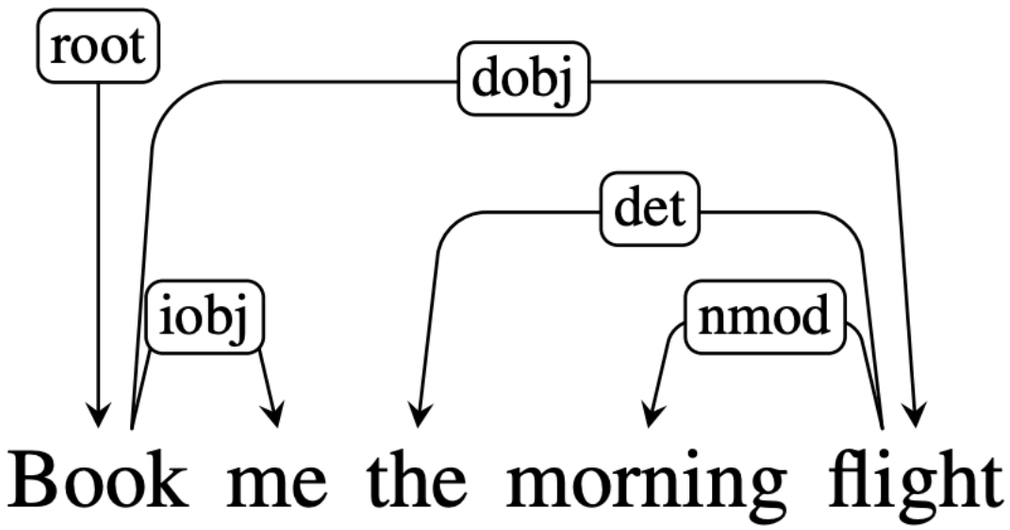
J&M Fig 15.5



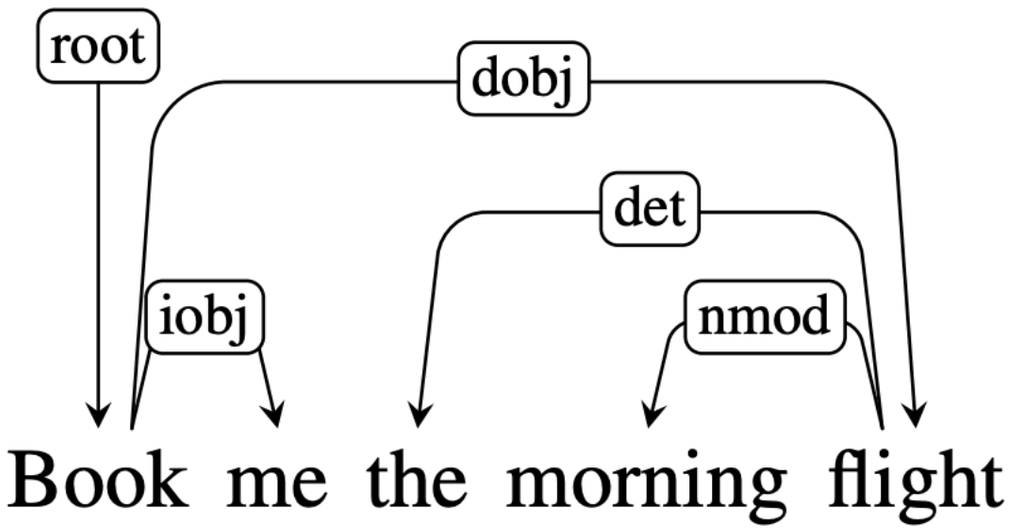
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	



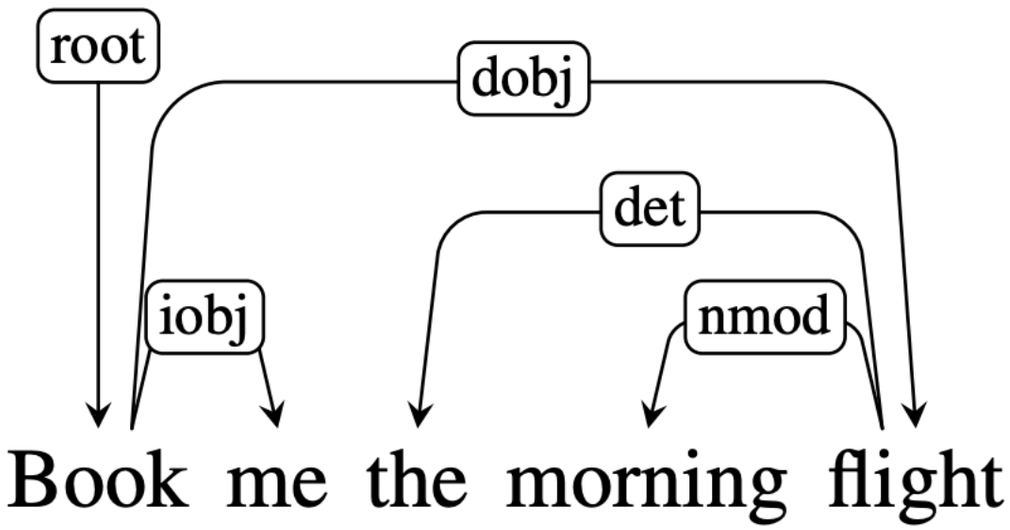
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	



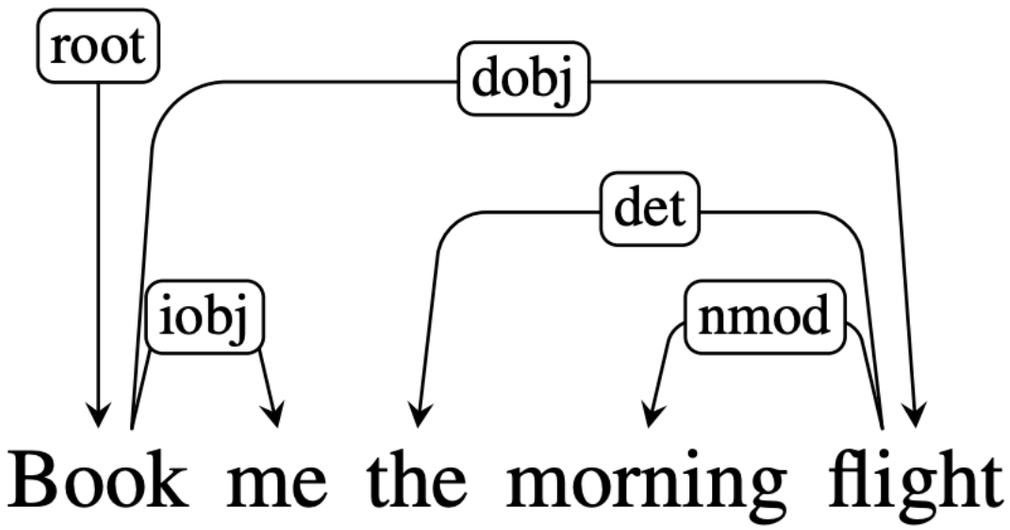
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)



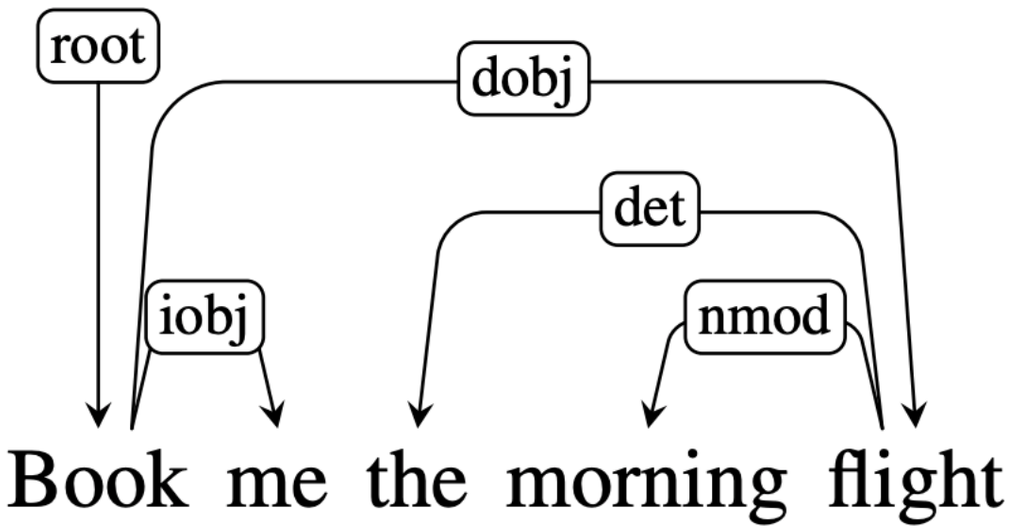
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	



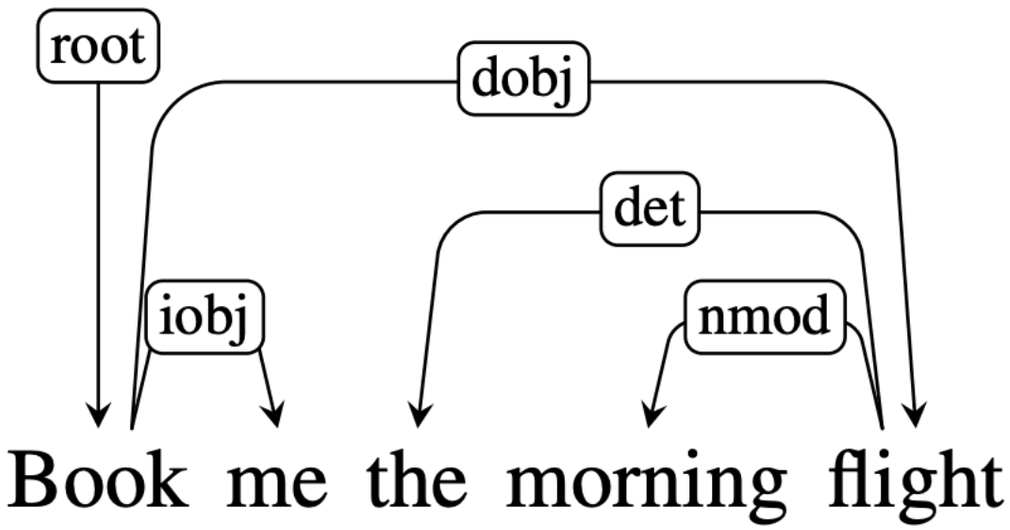
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)



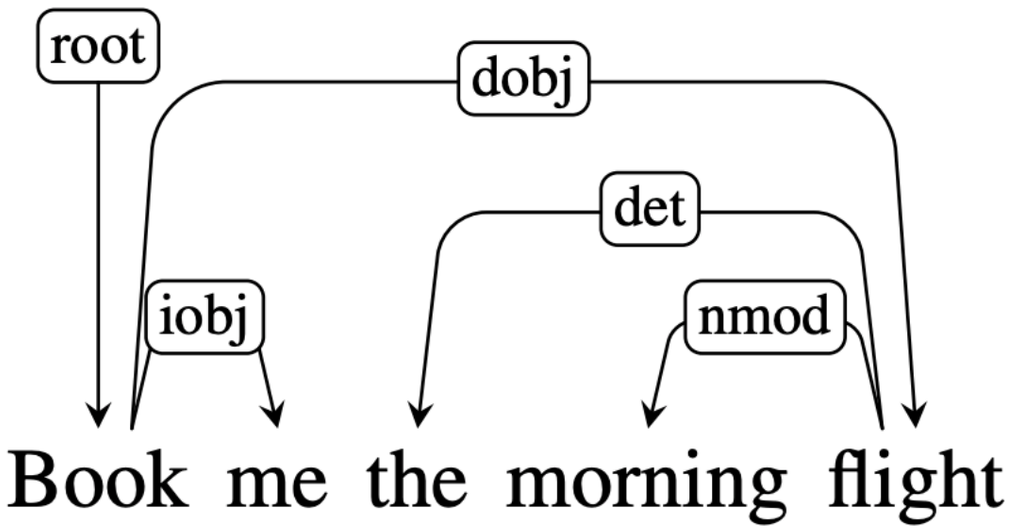
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)



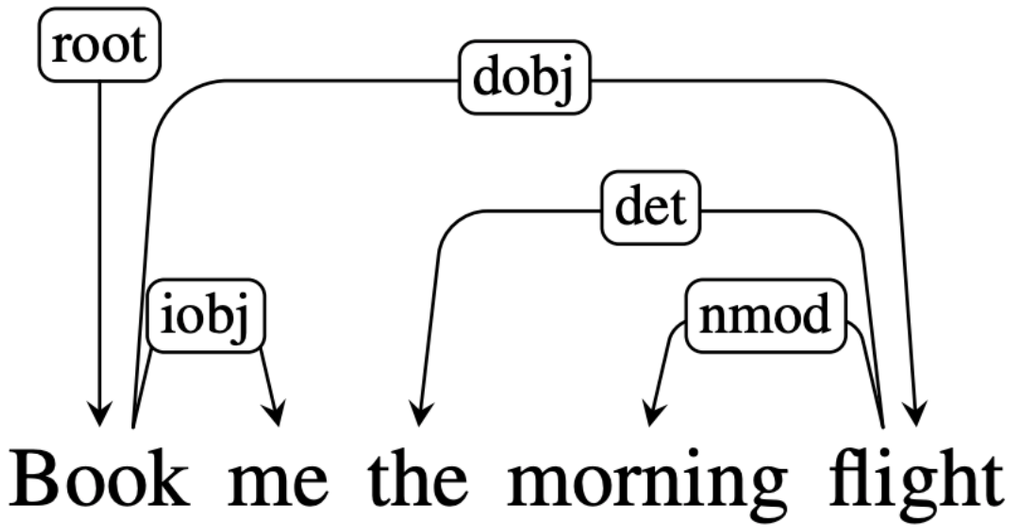
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Dependency Labels

- For simplicity, we omit labels on the dependency relations
- In practice, we parameterise the left-arc and right-arc actions with dependency labels:
  - E.g. left-arc-nsubj or right-arc-dobj
- This expands the list of actions to more than just 3 types

The Right Action?

- We assume an **oracle** that tells us the right action at every step
- Given a dependency tree, the role of oracle is to generate a sequence of ground truth actions



Action
SHIFT
SHIFT
RIGHTARC
SHIFT
SHIFT
SHIFT
LEFTARC
LEFTARC
RIGHTARC
RIGHTARC
Done

Parsing Model

- We then train a supervised model to **mimic** the actions of the oracle
  - To learn at every step the correct action to take (given by the oracle)
  - At test time, the trained model can be used to parse a sentence to create the dependency tree

Parsing As Classification

- Input:
  - Stack (top-2 elements: s_1 and s_2)
  - Buffer (first element: b_1)
- Output
  - 3 classes: *shift*, *left-arc*, or, *right-arc*
- Features
  - word (w), part-of-speech (t)

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United) (flights → morning) (flights → the)

- Input features:
  - $s_1.w$ = *flights*
  - $s_2.w$ = *cancelled*
  - $s_1.t$ = *NNS*
  - $s_2.t$ = *VBD*
  - $b_1.w$ = *to*
  - $b_1.t$ = *TO*
  - $s_1.t \circ s_2.t$ = *NNS_VBD*

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

- Output label: *shift*

Classifiers

- Traditionally SVM works best
- Nowadays, deep learning models are the state-of-the-art
- Weakness: local classifier based on greedy search
- Solutions:
  - Beam search: keep track of top-N best actions (next lecture!)
  - Dynamic oracle: during training, use predicted actions occasionally

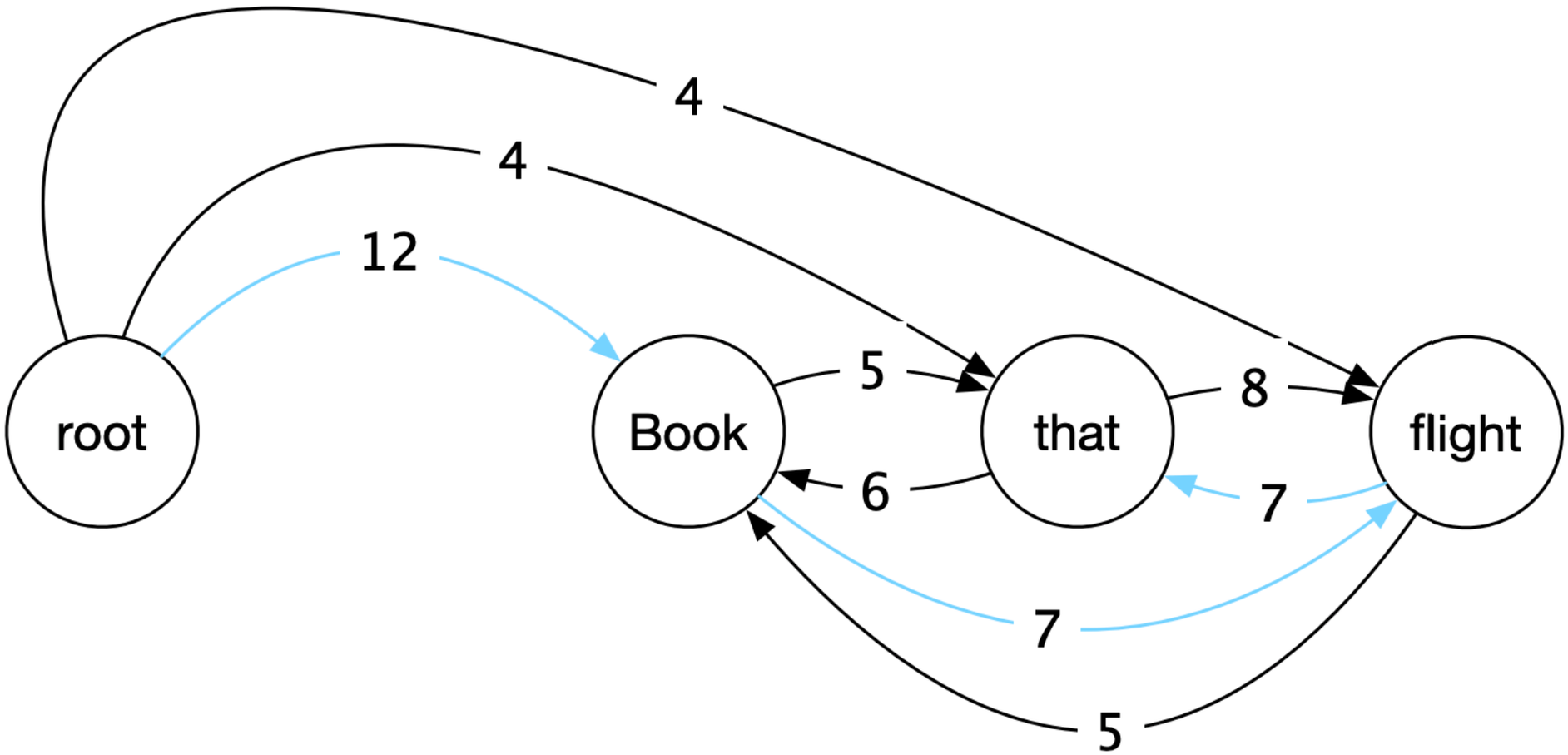
Graph-Based Parsing

- Given an input sentence, construct a fully-connected, weighted, directed graph
- Vertices: all words
- Edges: head-dependent arcs
- Weight: score based on training data (relation that is frequently observed receive a higher score)
- Objective: find the maximum spanning tree

Advantage

- Can produce non-projective trees
  - Not a big deal for English
  - But a problem for many other languages
- Score entire trees
  - Avoid making greedy local decisions like transition-based parsers
  - Captures long dependencies better

Example



- Caveat: tree may contain cycles
- Solution: need to do cleanup to remove cycles (Chu-Liu-Edmonds algorithm)

A Final Word

- Dependency parsing a compelling, alterative, formulation to constituency parsing
  - structures based on words as internal nodes
  - edges encode word-word syntactic and semantic relations
  - often this is the information we need for other tasks!
- Transition-based parsing
  - a sequence of shift and left/right-arc actions
- Graph-based parsing
  - frames it as a maximum spanning tree problem

Required Reading

- J&M3 Ch. 15