Syntactic and Dependency Parsing

Natalie Parde, Ph.D.

Department of Computer Science

University of Illinois at Chicago

CS 421: Natural Language Processing

Fall 2019

Many slides adapted from Jurafsky and Martin (https://web.stanford.edu/~jurafsky/slp3/) and Ben-Gurion University's NLP course (https://www.cs.bgu.ac.il/~michaluz/seminar/CKY1.pdf).

What is syntactic parsing?

The process of automatically recognizing sentences and assigning syntactic (grammatical) structure to them.

Why is syntactic parsing useful?

- Lots of reasons!
  - Grammar checking
    - Sentences that can't be parsed may be grammatically incorrect (or at least hard to read)
  - Semantic analysis
  - Downstream applications
    - Question answering
    - Information extraction

What courses were **taught by UIC CS assistant professors in 2019**?

Subject = courses …don't return a list of UIC CS assistant professors!

Parsing algorithms are one of the core tools for analyzing natural language.

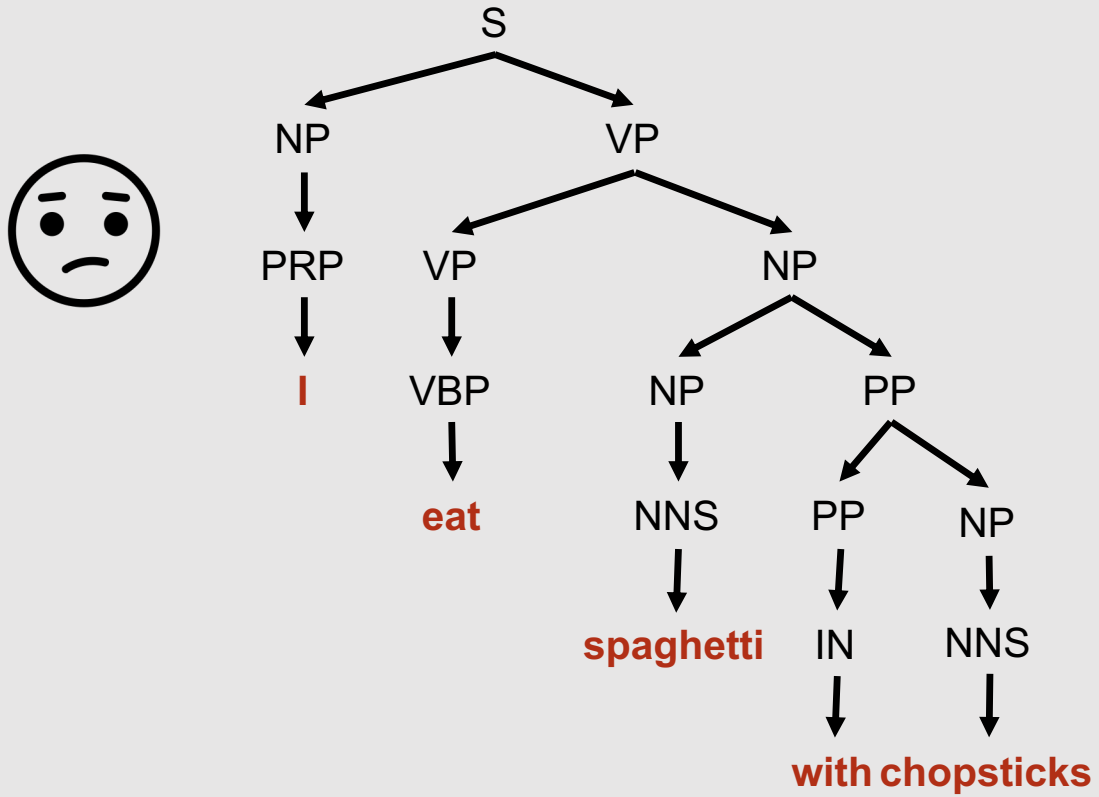
- Parsing algorithms automatically describe the syntactic structure of sentences in terms of **context-free grammars** (such as those discussed last week)
- This can be viewed as a **search problem**:
  - Given the set of all possible parse trees, find the correct parse tree for this sentence.

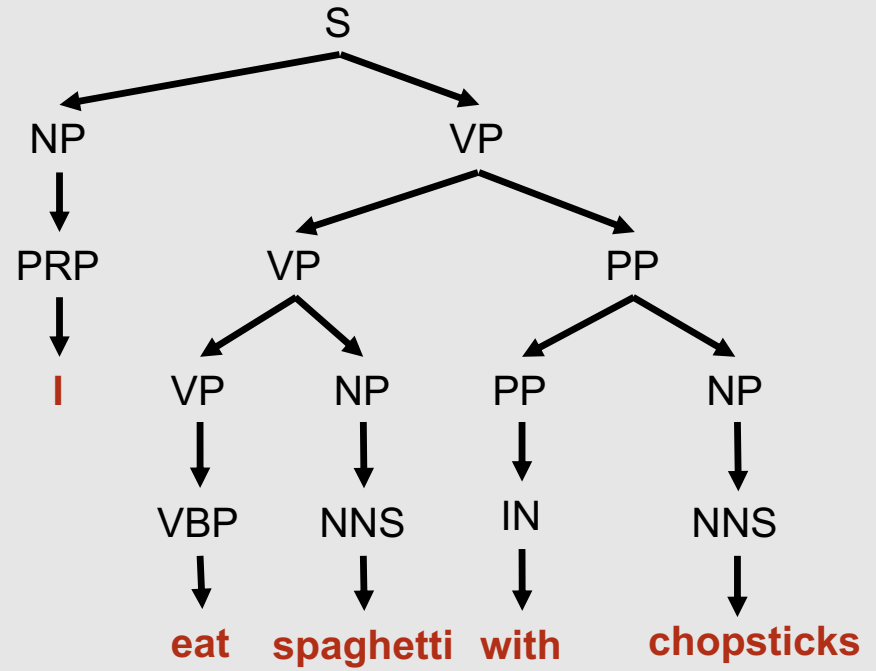
Recognition vs. Parsing

- **Recognition:** Deciding whether a sentence belongs to the language specified by a formal grammar.
- **Parsing:** Producing a parse tree for the sentence based on that formal grammar.
- Both tasks are necessary for generating correct syntactic parses!
  - Failure to accurately recognize whether a sentence can be parsed will lead to **misparses**, which will in turn lead to additional errors in downstream applications.
- Parsing is more “difficult” (greater time complexity) than recognition

Remember, language is ambiguous!

Input sentences may have many possible parses





There are many ways to generate parse trees.

Top-Down Parsing:

- Goal-driven
- Builds parse tree from the start symbol down to the terminal nodes

Bottom-Up Parsing:

- Data-driven
- Builds parse tree from the terminal nodes up to the start symbol

Naïve approach: Enumerate all possible solutions

Dynamic programming approach: Save partial solutions in a table, and use this information to reduce search time

These approaches can be implemented naïvely, or using more advanced techniques.

Top-Down Parsing

- Assume that the input can be derived by the designated start symbol ***S***
- Find the tops of all trees that can start with ***S***
  - Look for all production rules with ***S*** on the left-hand side
- Find the tops of all trees that can start with those constituents
- (Repeat recursively until POS categories at bottom of tree are reached)
- Trees whose leaves fail to match all words in the input sentence can be rejected, leaving behind trees that represent successful parses

Top-Down Parsing: Example

Input Sentence:

Book that flight.

Grammar:

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Lexicon:

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

Top-Down Parsing: Example

Book that flight.

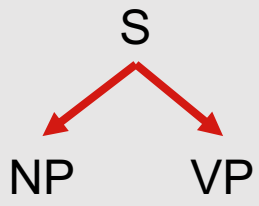
S S S

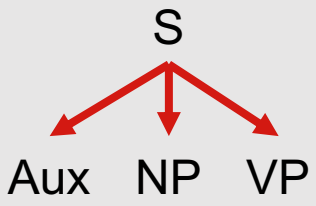
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Top-Down Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

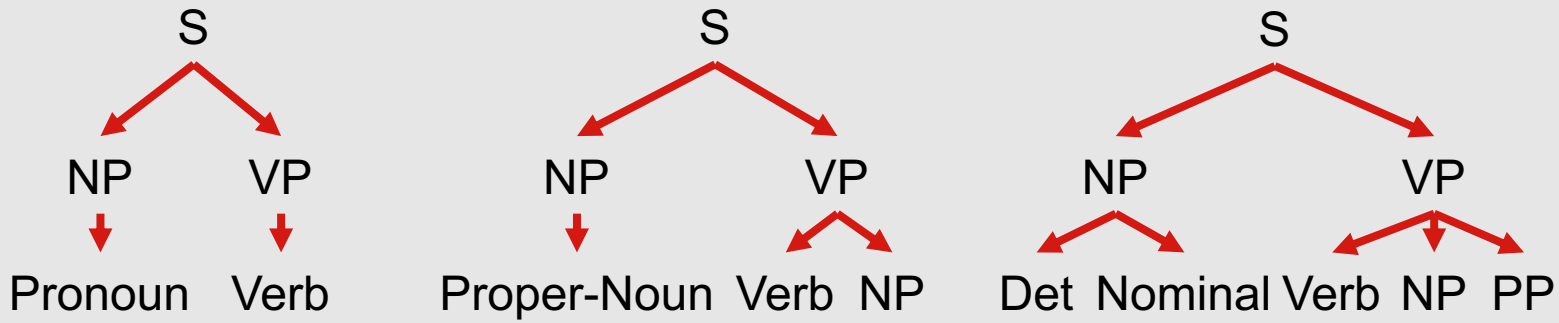


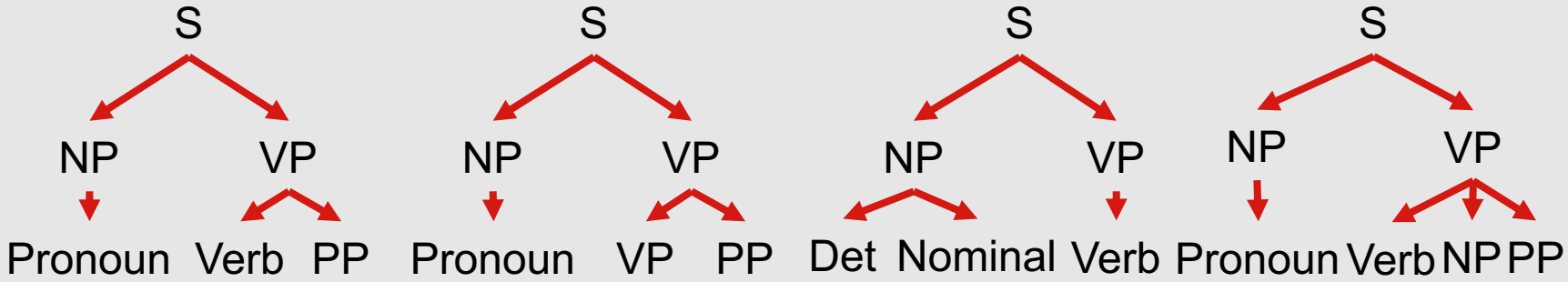


Top-Down Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



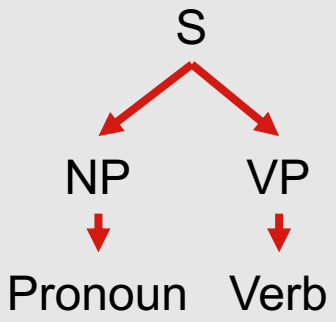


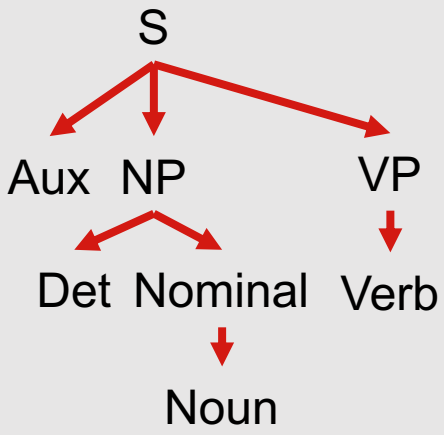
…and many more!

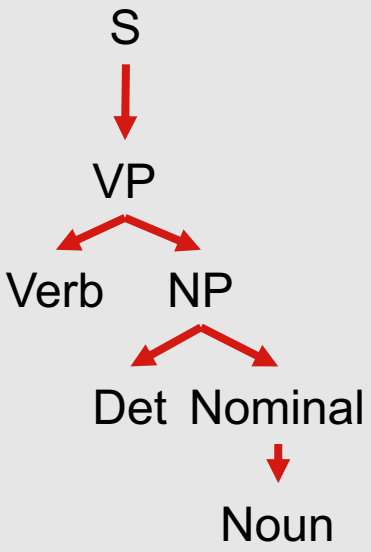
Top-Down Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP





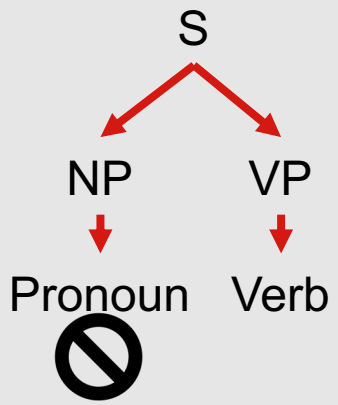


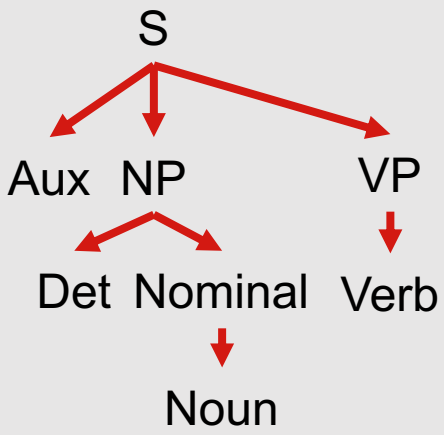
…and many, many more not shown!

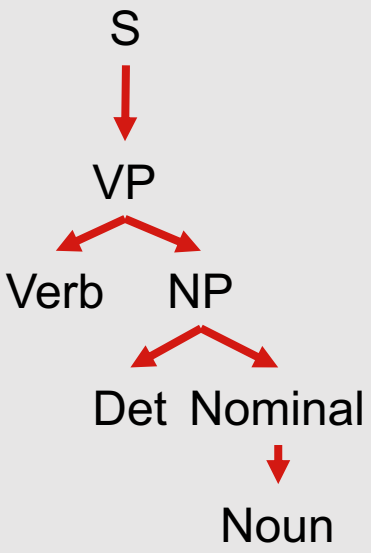
Top-Down Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP







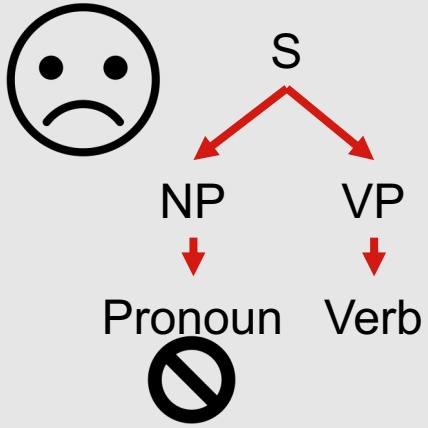
Det → that | this | a
Noun → book | flight | meal | money
Verb → **book** | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

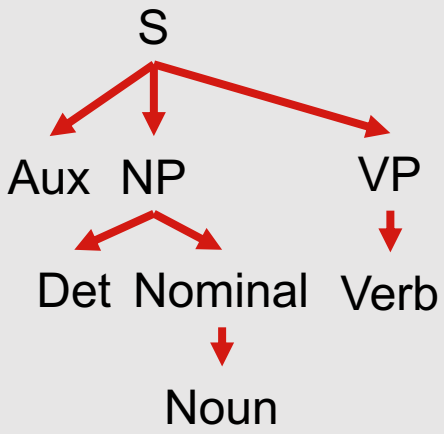
…and many, many more not shown!

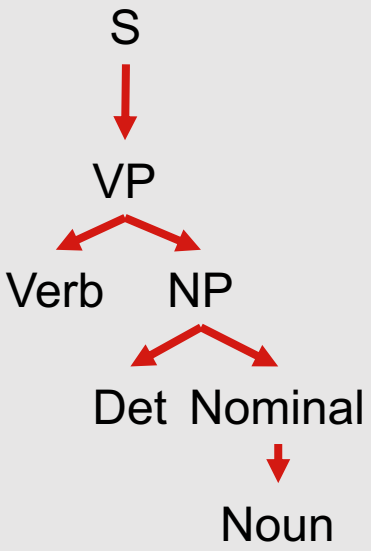
Top-Down Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP







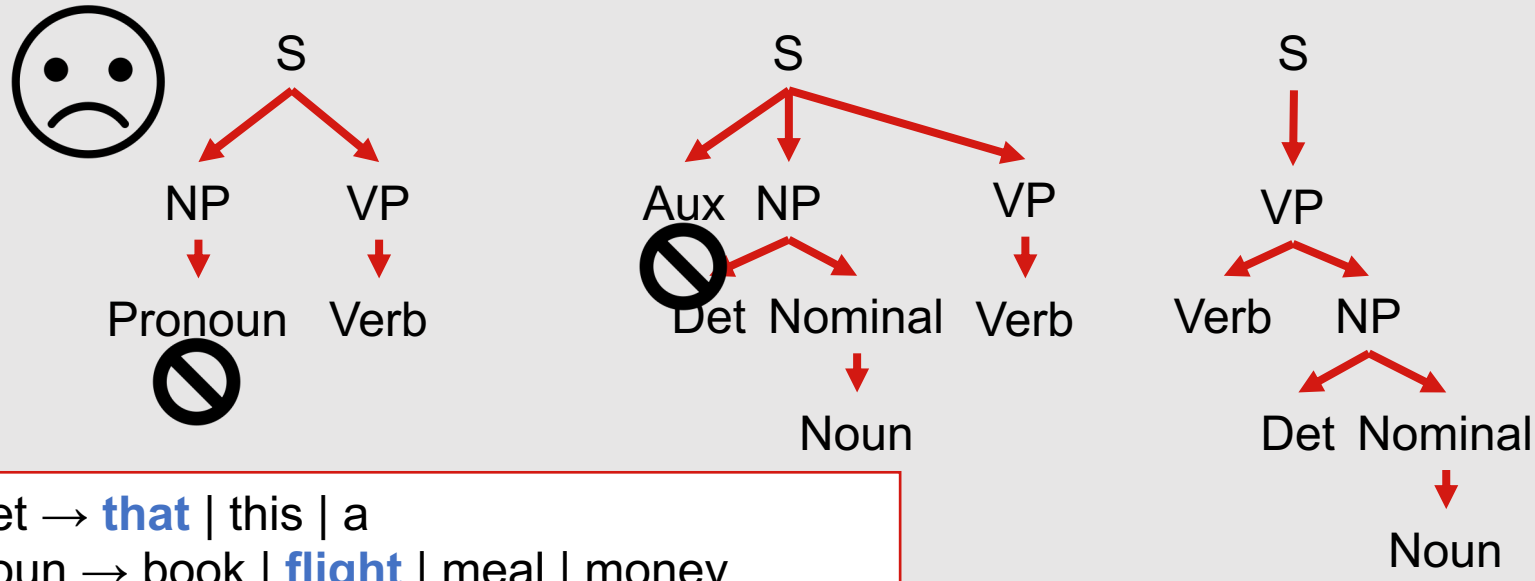
Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

…and many, many more not shown!

Top-Down Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



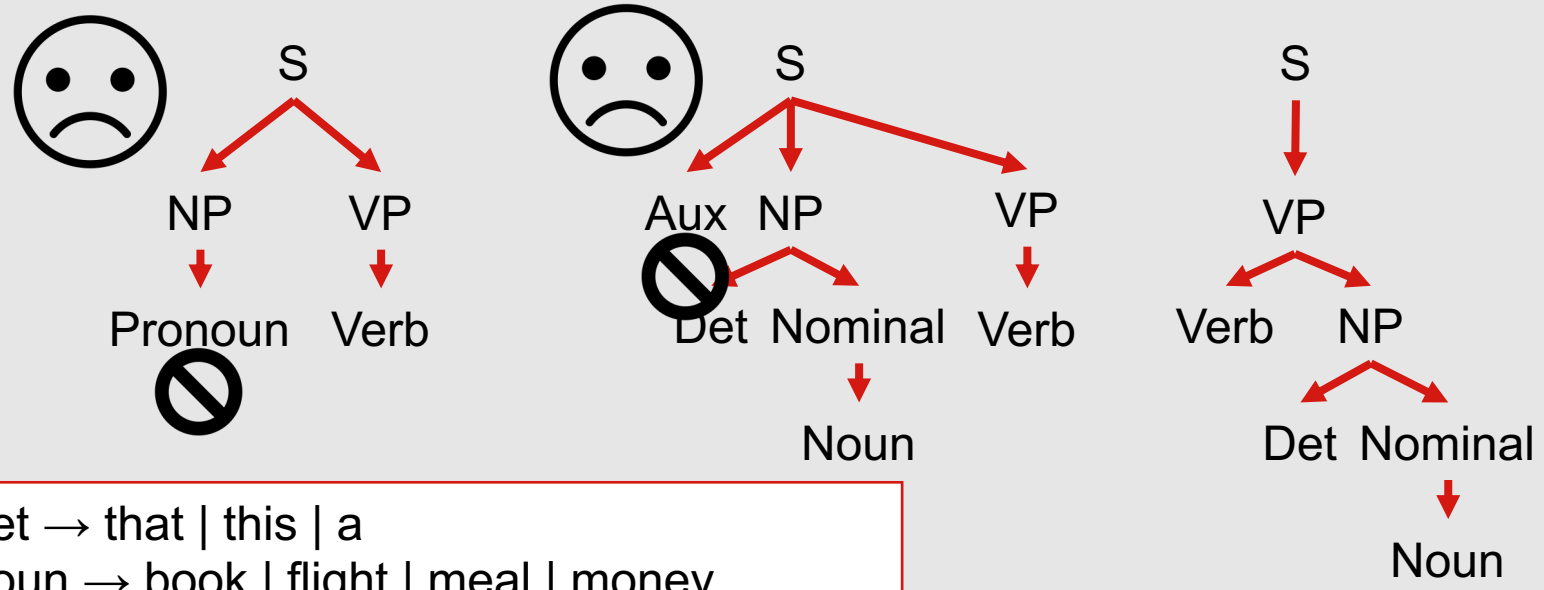
Det → **that** | this | a
Noun → book | **flight** | meal | money
Verb → **book** | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

…and many, many more not shown!

Top-Down Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



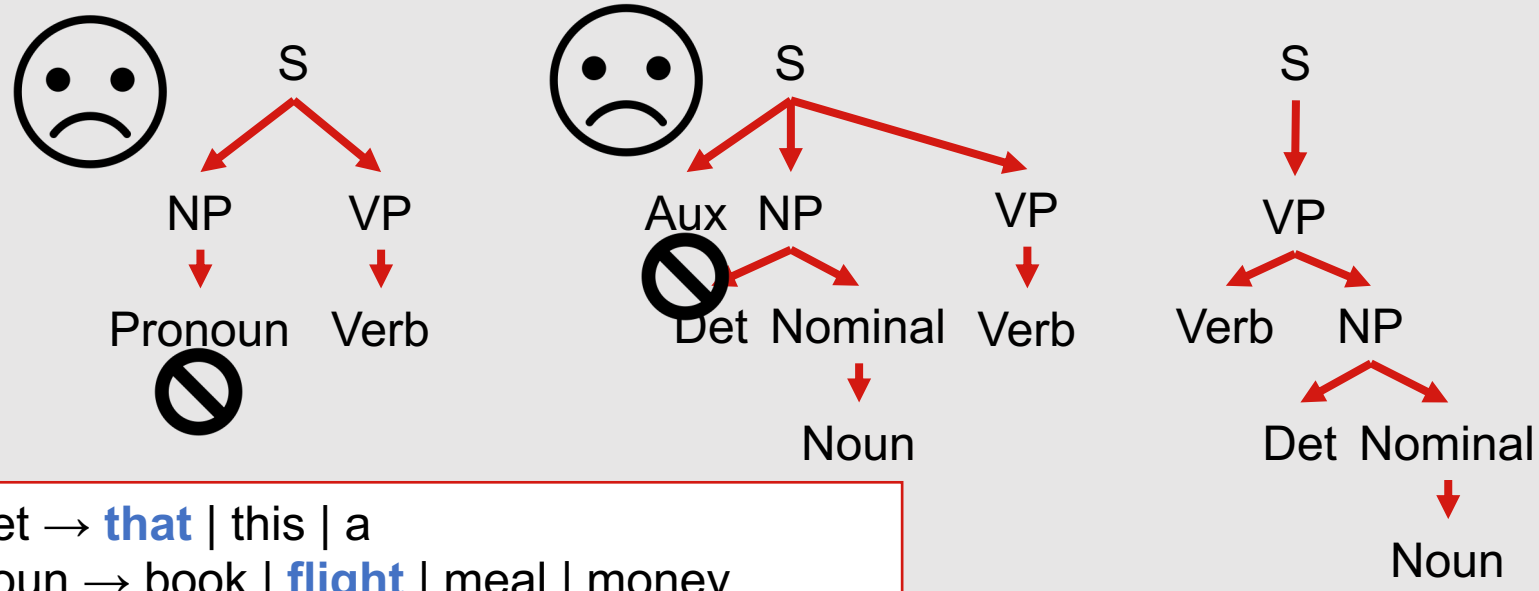
Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

…and many, many more not shown!

Top-Down Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



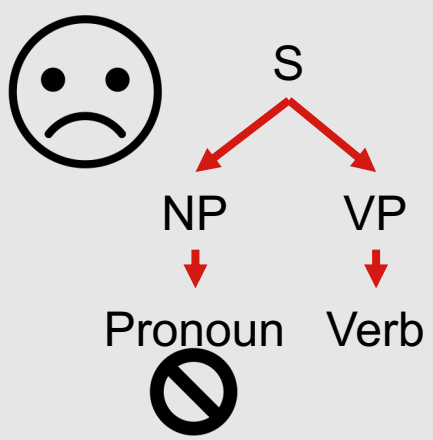
Det → **that** | this | a
Noun → book | **flight** | meal | money
Verb → **book** | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

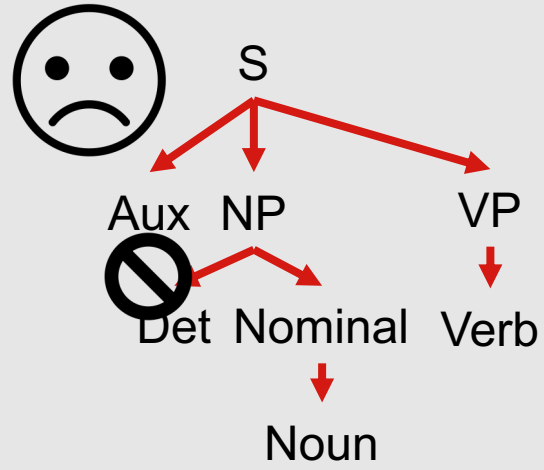
…and many, many more not shown!

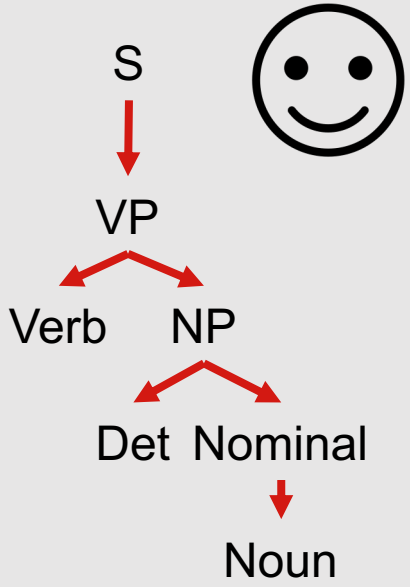
Top-Down Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP







Det → **that** | this | a
Noun → book | **flight** | meal | money
Verb → **book** | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

…and many, many more not shown!

Bottom-Up Parsing

- Earliest known parsing algorithm!
- Starts with the words in the input sentence, and tries to build trees from those words up by applying rules from the grammar one at a time
  - Looks for places in the in-progress parse where the righthand side of a production rule might fit
- Success = parser builds a tree rooted in the start symbol ***S*** that covers all of the input words

Bottom-Up Parsing: Example

Input Sentence:

Book that flight.

Grammar:

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Lexicon:

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

Bottom-Up Parsing: Example

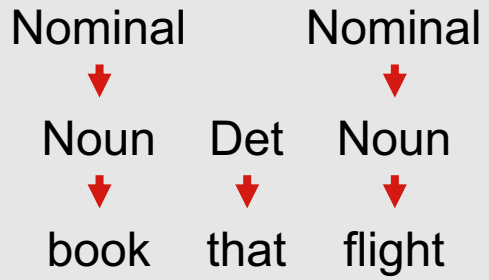
Book that flight.

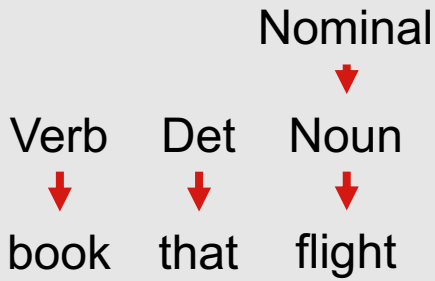
Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

Bottom-Up Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

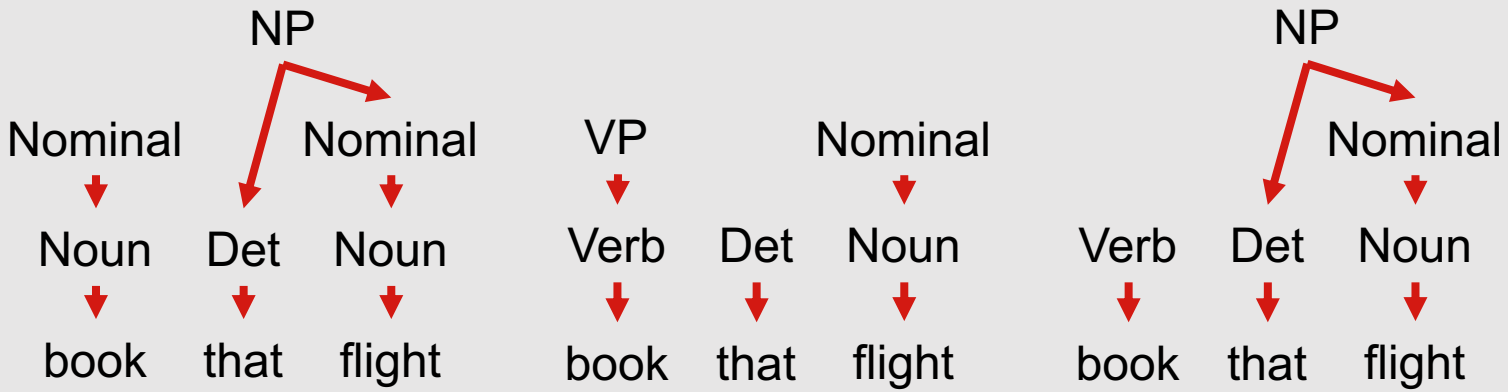




Bottom-Up Parsing: Example

Book that flight.

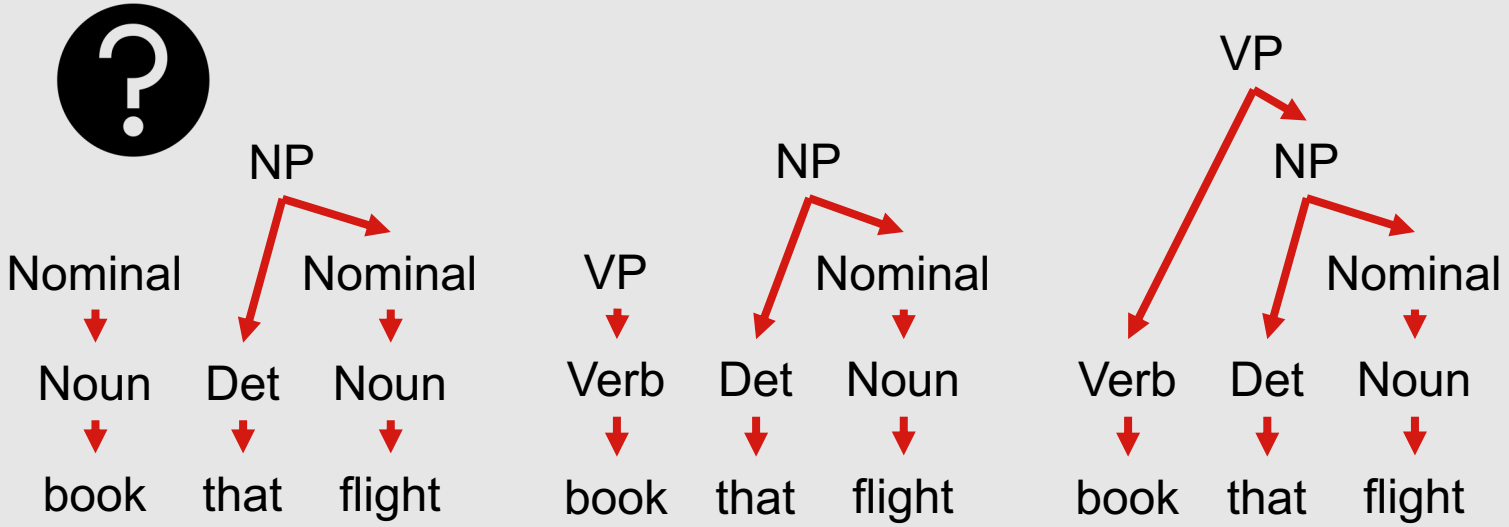
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



Bottom-Up Parsing: Example

Book that flight.

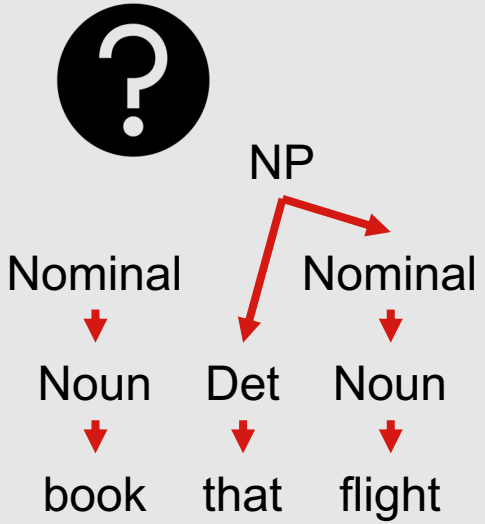
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

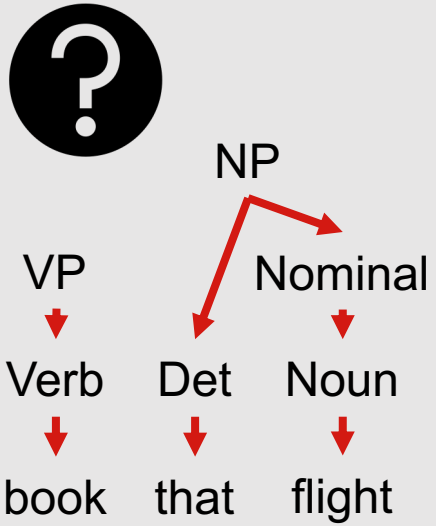


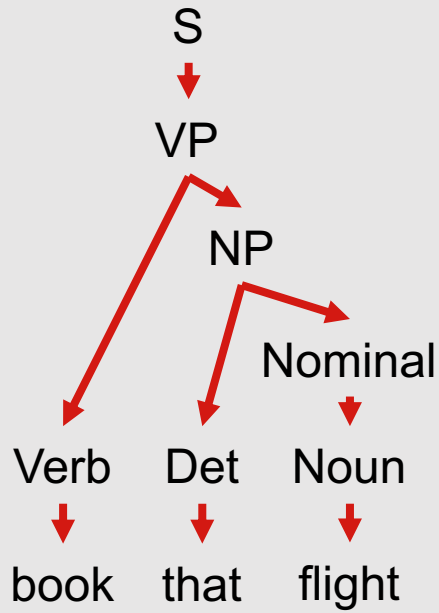
Bottom-Up Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP



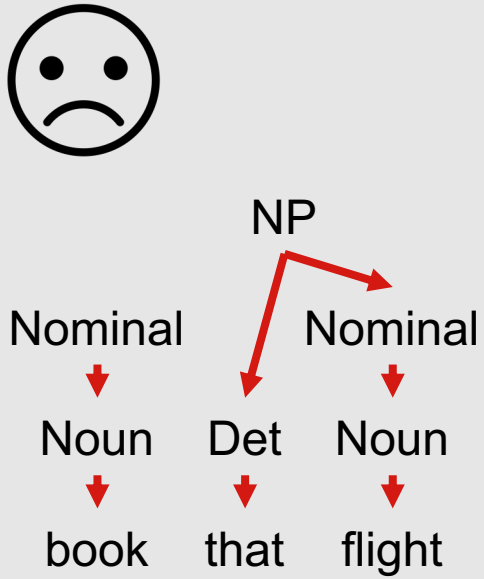


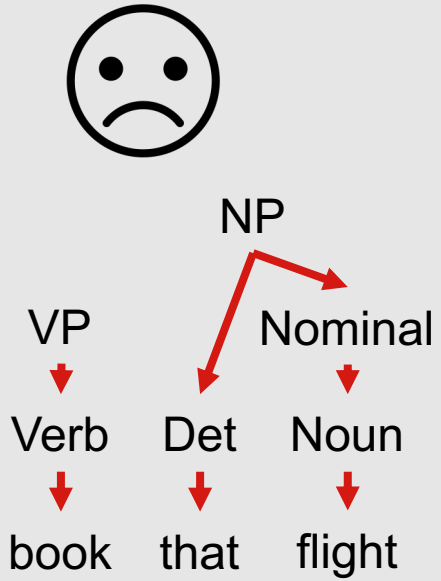


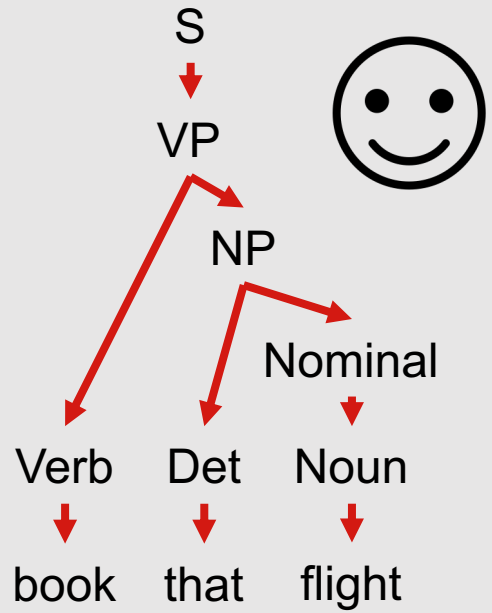
Bottom-Up Parsing: Example

Book that flight.

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP







Top-Down vs. Bottom-Up Parsing

Top-Down Parsing

- Pros:
  - Never wastes time exploring trees that cannot result in a sentence
  - Never explores subtrees that cannot fit into a larger valid (i.e., results in a sentence) tree
- Cons:
  - Spends considerable effort on trees that are not consistent with the input

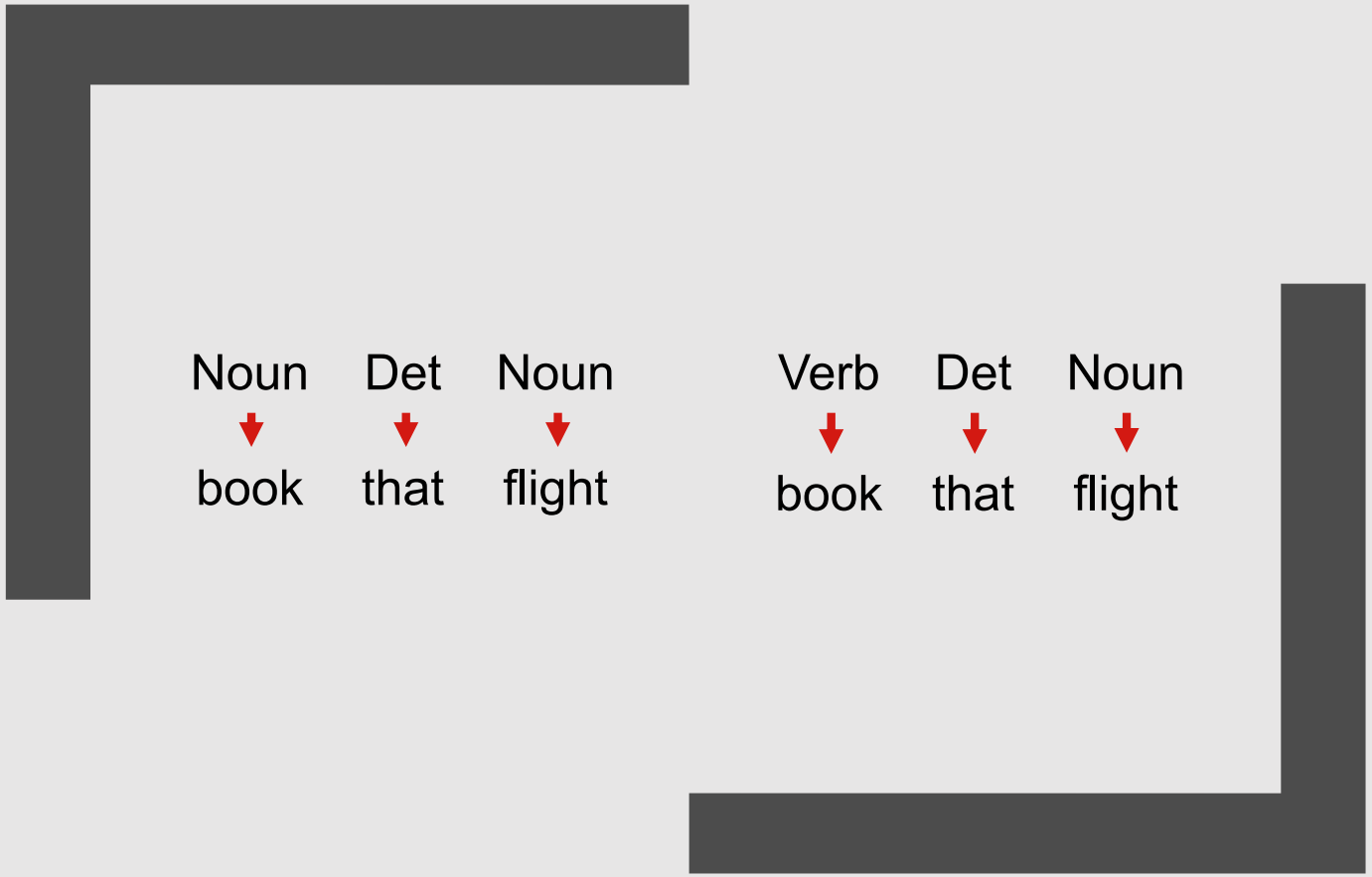
Bottom-Up Parsing

- Pros:
  - Never suggests trees that are inconsistent with the input
- Cons:
  - Generates many trees and subtrees that cannot result in a valid sentence (according to production rules specified by the grammar)

More about ambiguity….

- **Structural Ambiguity:** Occurs when a grammar allows for more than one possible parse for a given sentence
- Two Forms:
  - **Attachment Ambiguity:** Occurs when a constituent can be attached to a parse tree at more than one place
    - I eat spaghetti **with chopsticks**.
  - **Coordination Ambiguity:** Occurs when different sets of phrases can be conjoined by a conjunction
    - I grabbed a muffin from the table marked "nut-free scones **and** muffins," hoping I'd parsed the sign correctly.

Remember …local ambiguity can also exist.



- Det → that | this | a
- Noun → **book** | flight | meal | money
- Verb → **book** | include | prefer
- Pronoun → I | she | me
- Proper-Noun → Houston | NWA
- Aux → does
- Preposition → from | to | on | near | through

All of this ambiguity can lead to really complex search spaces!

- **Backtracking** approaches expand the search space incrementally, systematically exploring one state at a time
- When they arrive at trees inconsistent with the input, they return to an unexplored alternative
- However, in doing so they tend to discard valid subtrees …this means that time-consuming work needs to be repeated
- More efficient approach?
  - **Dynamic programming**

Dynamic Programming Parsing Methods

- **Tables store subtrees for constituents as they are discovered**
- Solves:
  - Re-parsing problem
  - (Partially) ambiguity problem, since the table implicitly stores all possible parses

Dynamic Programming Parsing Methods

- Most widely used methods:
  - Cocke-Kasami-Younger (**CKY**) algorithm
  - **Earley** algorithm
  - **Chart parsing**

CKY Algorithm

- One of the earliest recognition and parsing algorithms
- **Bottom-up dynamic programming**
- Standard version can only recognize CFGs in **Chomsky Normal Form** (CNF)

Chomsky Normal Form

- Grammars are restricted to production rules of the form:
  - $A \rightarrow B\ C$
  - $A \rightarrow w$
- This means that the righthand side of each rule must expand to either two non-terminals or a single terminal
- Any CFG can be converted to a corresponding CNF grammar that accepts exactly the same set of strings as the original grammar!

How does this conversion work?

- Three situations we need to address:
  1. Production rules that mix terminals and non-terminals on the righthand side
  2. Production rules that have a single non-terminal on the righthand side (**unit productions**)
  3. Production rules that have more than two non-terminals on the righthand side
- Situation #1: **Introduce a dummy non-terminal that covers only the original terminal**
  - INF-VP → to VP could be replaced with INF-VP → TO VP and TO → to
- Situation #2: **Replace the non-terminals with the non-unit production rules to which they eventually lead**
  - A → B and B → w could be replaced with A → w
- Situation #3: **Introduce new non-terminals that spread longer sequences over multiple rules**
  - A → B C D could be replaced with A → B X1 and X1 → C D

CNF Conversion: Example

- S → NP VP
- S → Aux NP VP
- S → VP
- NP → Pronoun
- NP → Proper-Noun
- NP → Det Nominal
- Nominal → Noun
- Nominal → Nominal Noun
- Nominal → Nominal PP
- VP → Verb
- VP → Verb NP
- VP → Verb NP PP
- VP → Verb PP
- VP → VP PP
- PP → Preposition NP

Original	CNF
S → NP VP	S → NP VP
S → Aux NP VP	S → X1 VP
	X1 → Aux NP
S → VP	S → book | include | prefer
	S → Verb NP
	S → X2 PP
	X2 → Verb NP
	S → Verb PP
	S → VP PP

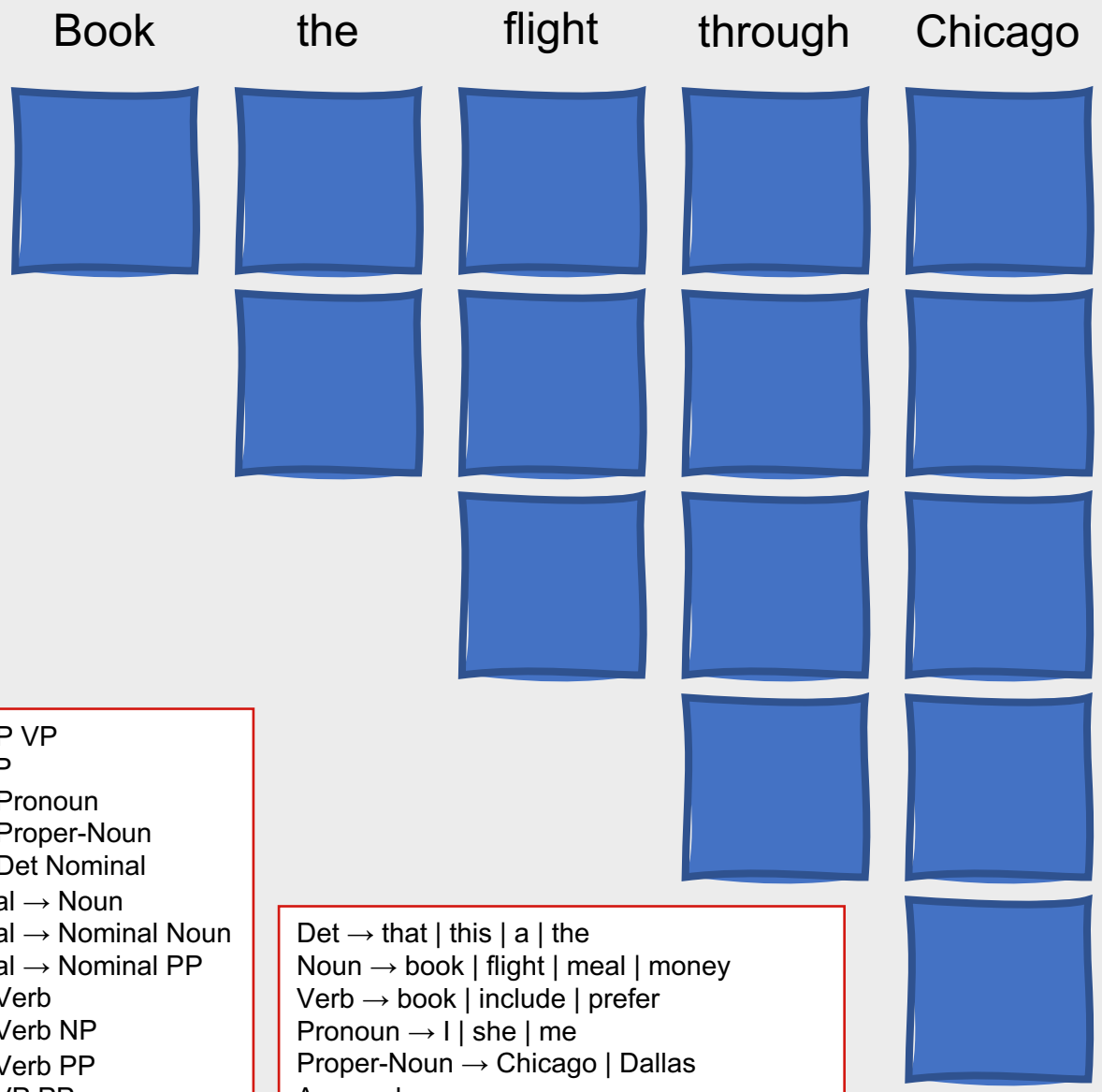
CKY Algorithm

- With the grammar in CNF, each non-terminal node above the POS level of the parse tree will have exactly two children
- Thus, a two-dimensional matrix can be used to encode the tree structure
- For sentence of length *n*, work with upper-triangular portion of (*n*+1) x (*n*+1) matrix
- Each cell [*i*,*j*] contains a set of non-terminals that represent all constituents spanning positions *i* through *j* of the input
  - Cell that represents the entire input resides in position [0,*n*]

CKY Algorithm

- Non-terminal entries: For each constituent [*i*,*j*], there is a position, *k*, where the constituent can be split into two parts such that $i < k < j$
  - [*i*,*k*] must lie to the left of [*i*,*j*] somewhere along row *i*, and [*k*,*j*] must lie beneath it along column *j*
- To fill in the parse table, we proceed in a bottom-up fashion so when we fill a cell [*i*,*j*], the cells containing the parts that could contribute to this entry have already been filled

CKY Algorithm: Example



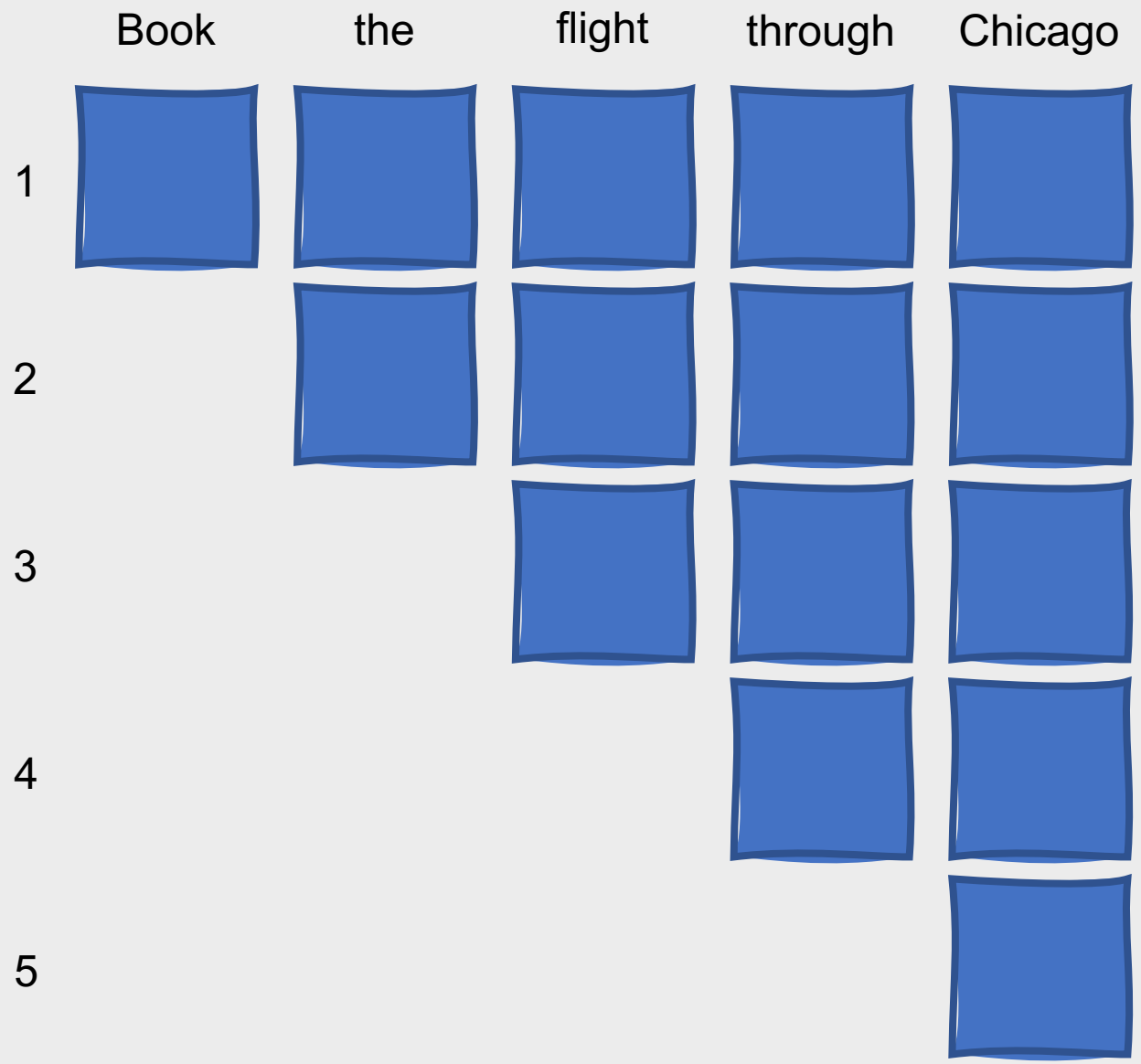
S → NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

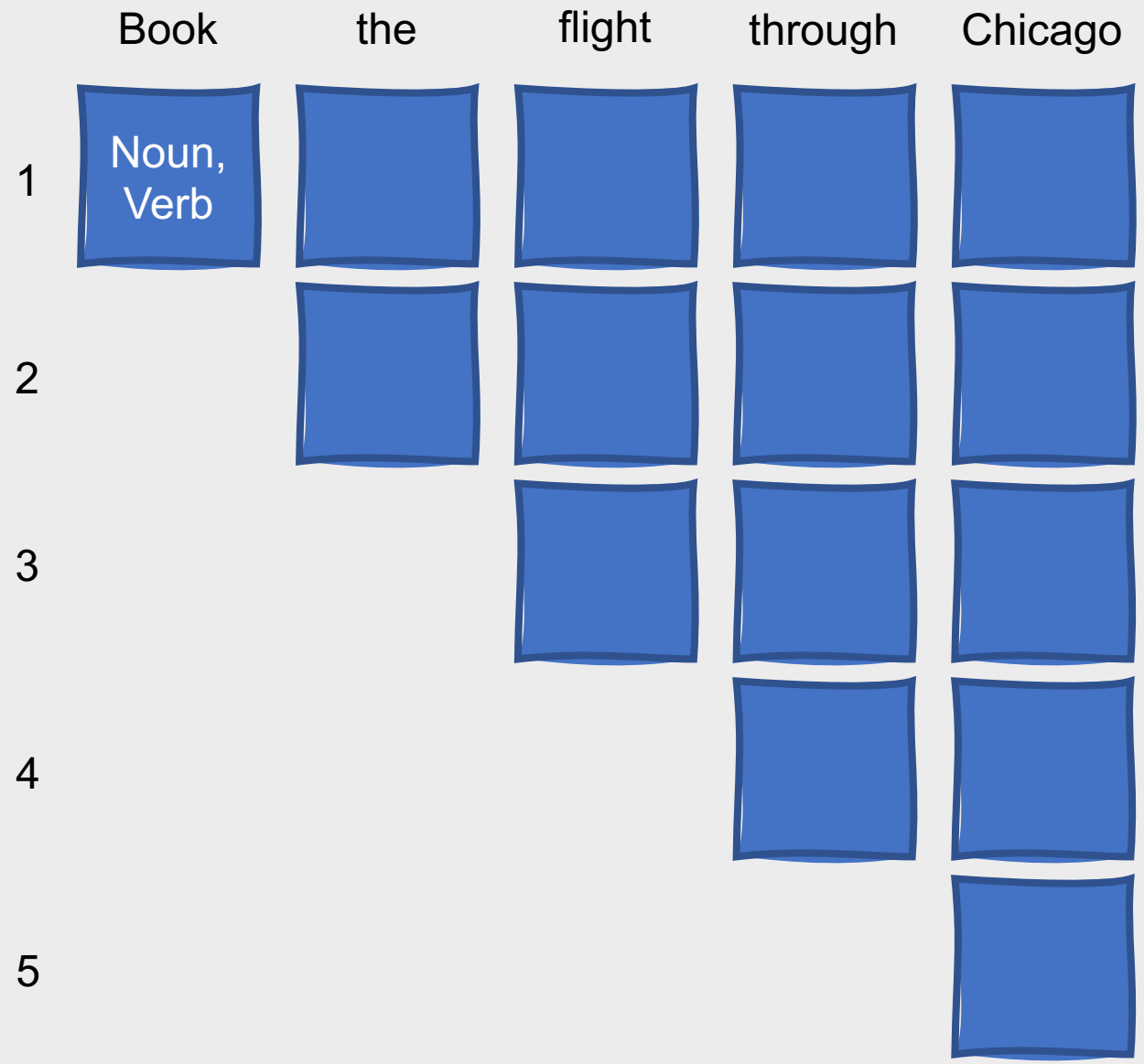
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → **book** | flight | meal | money
Verb → **book** | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

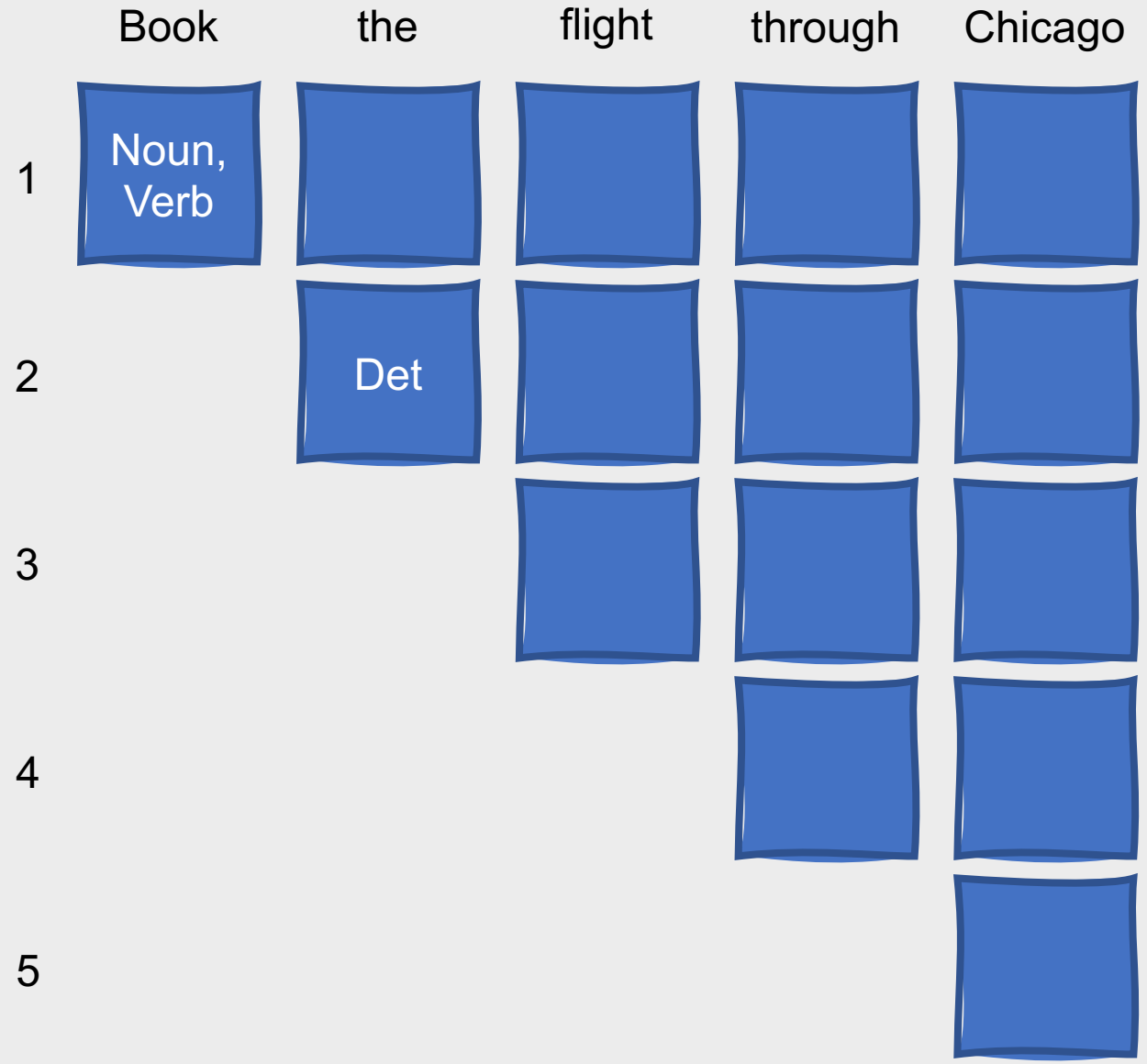
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | **the**
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

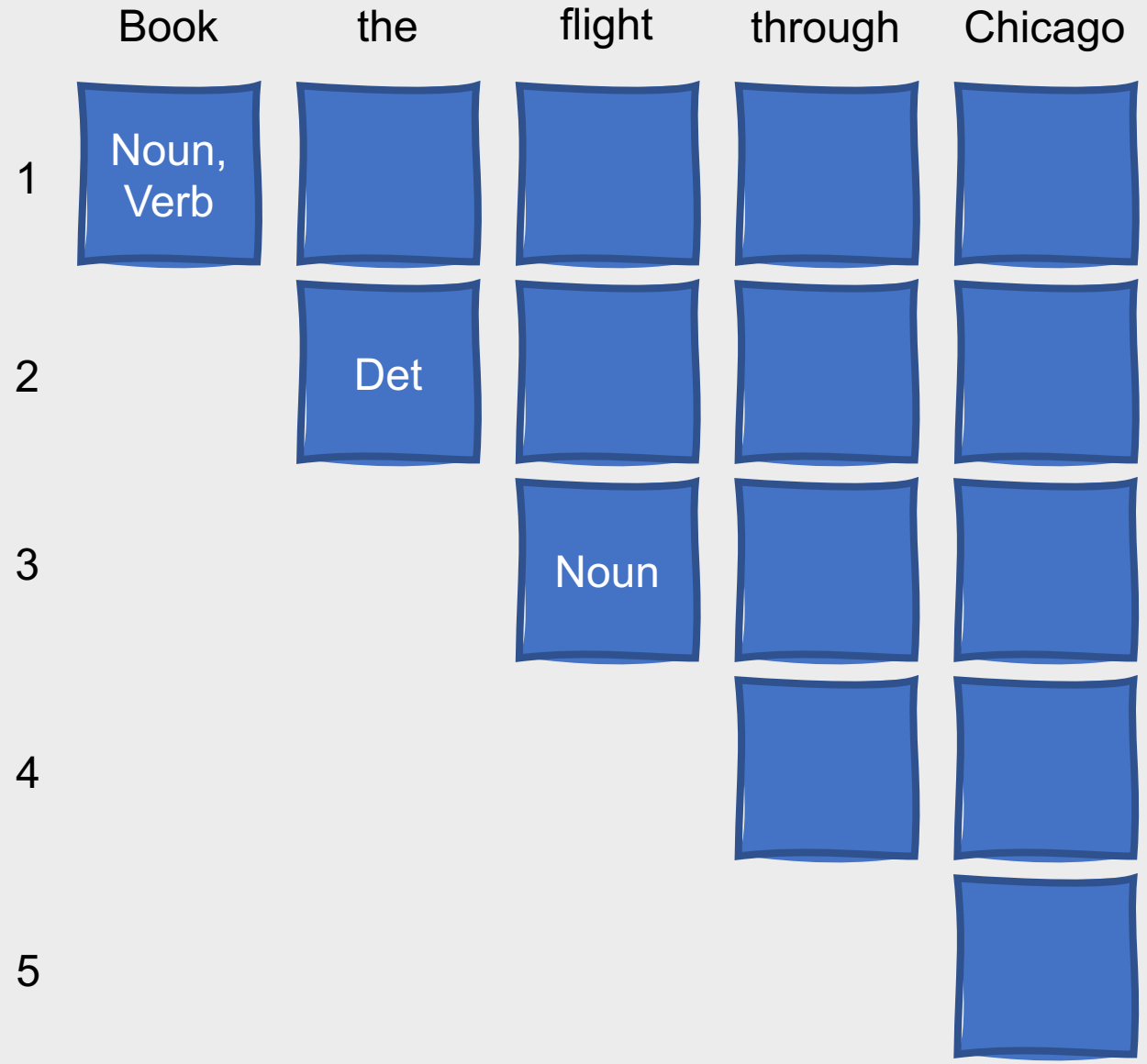
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | **flight** | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

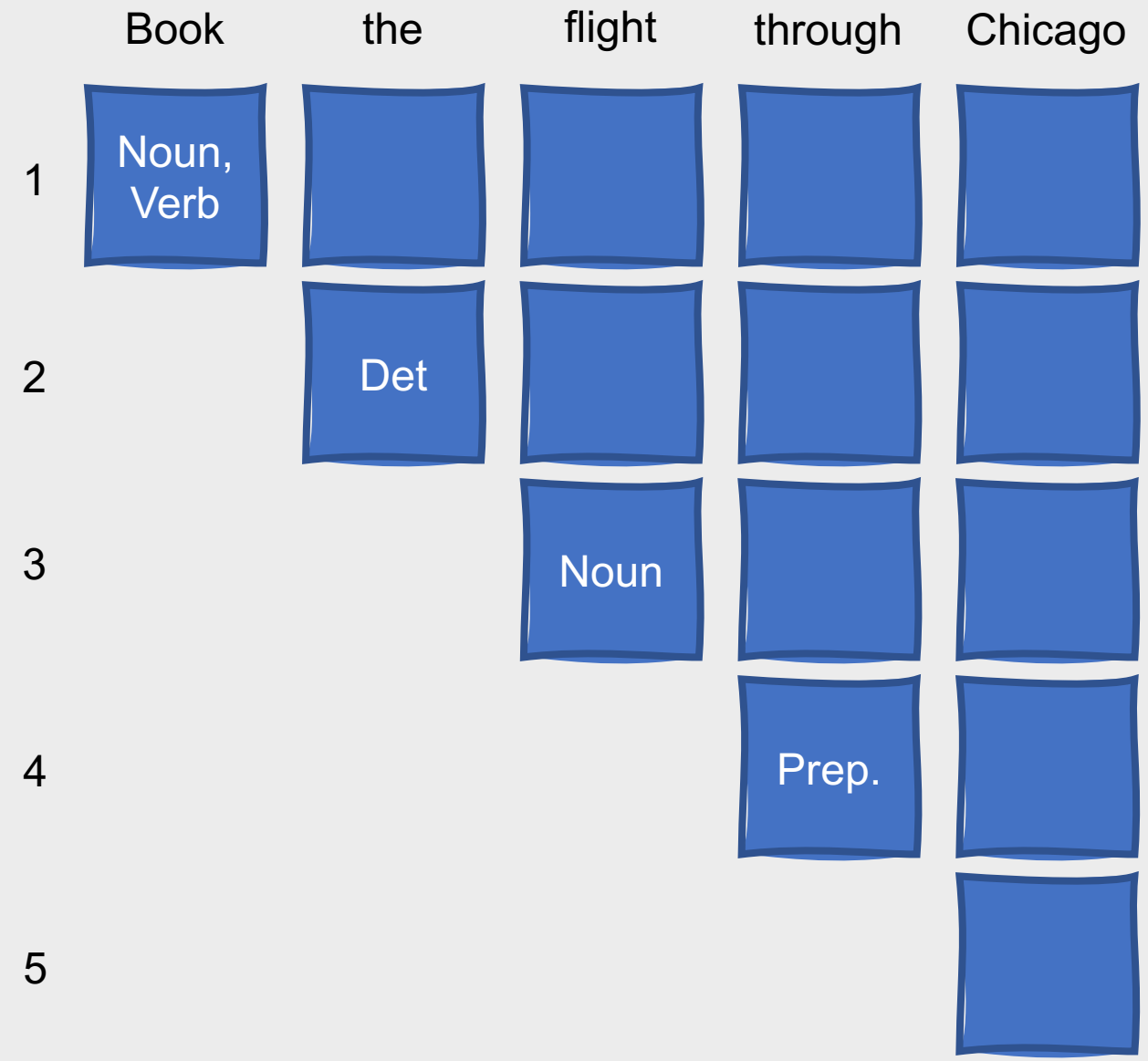
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | **through**

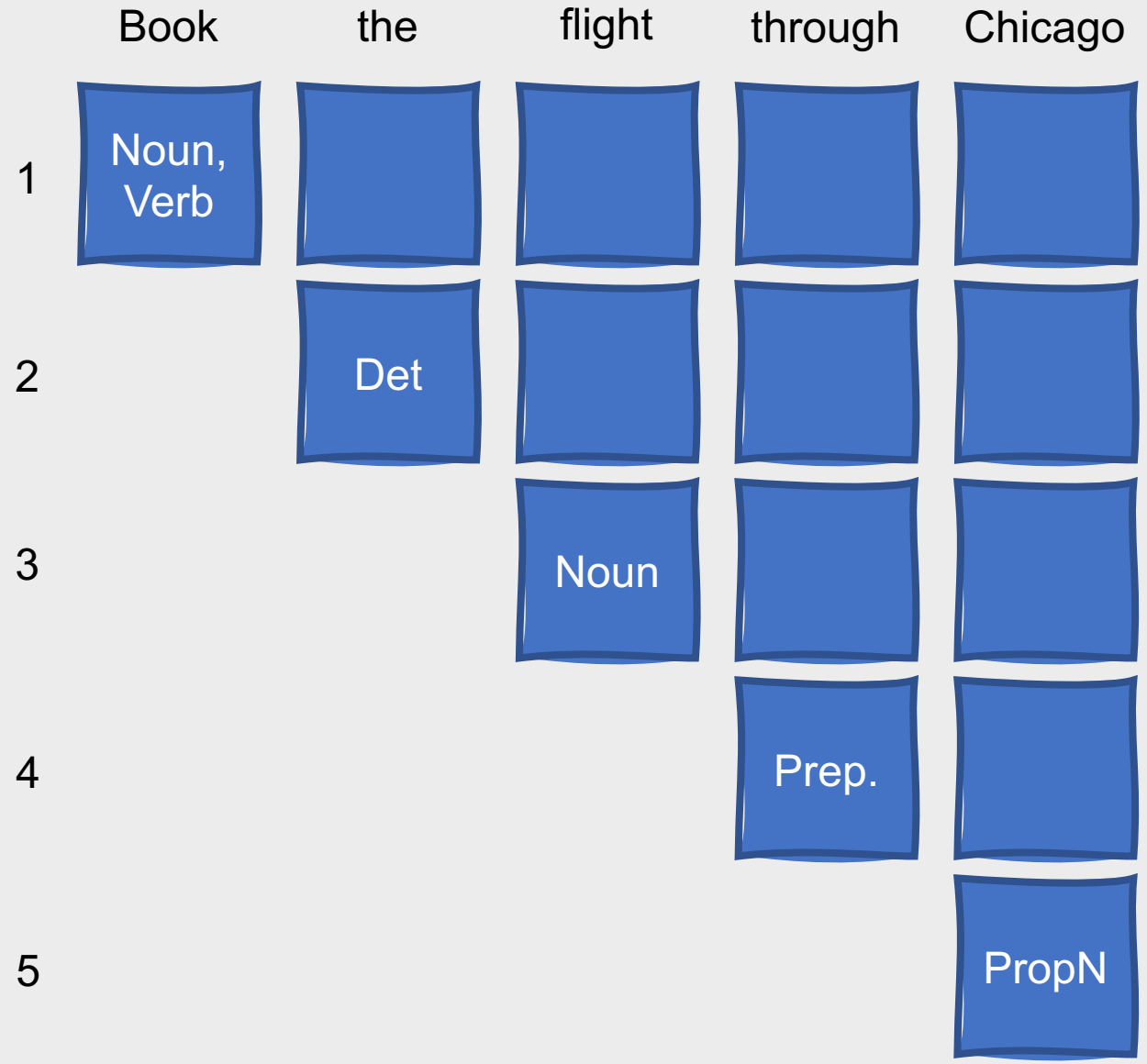
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → **Chicago** | Dallas
Aux → does
Preposition → from | to | on | near | through

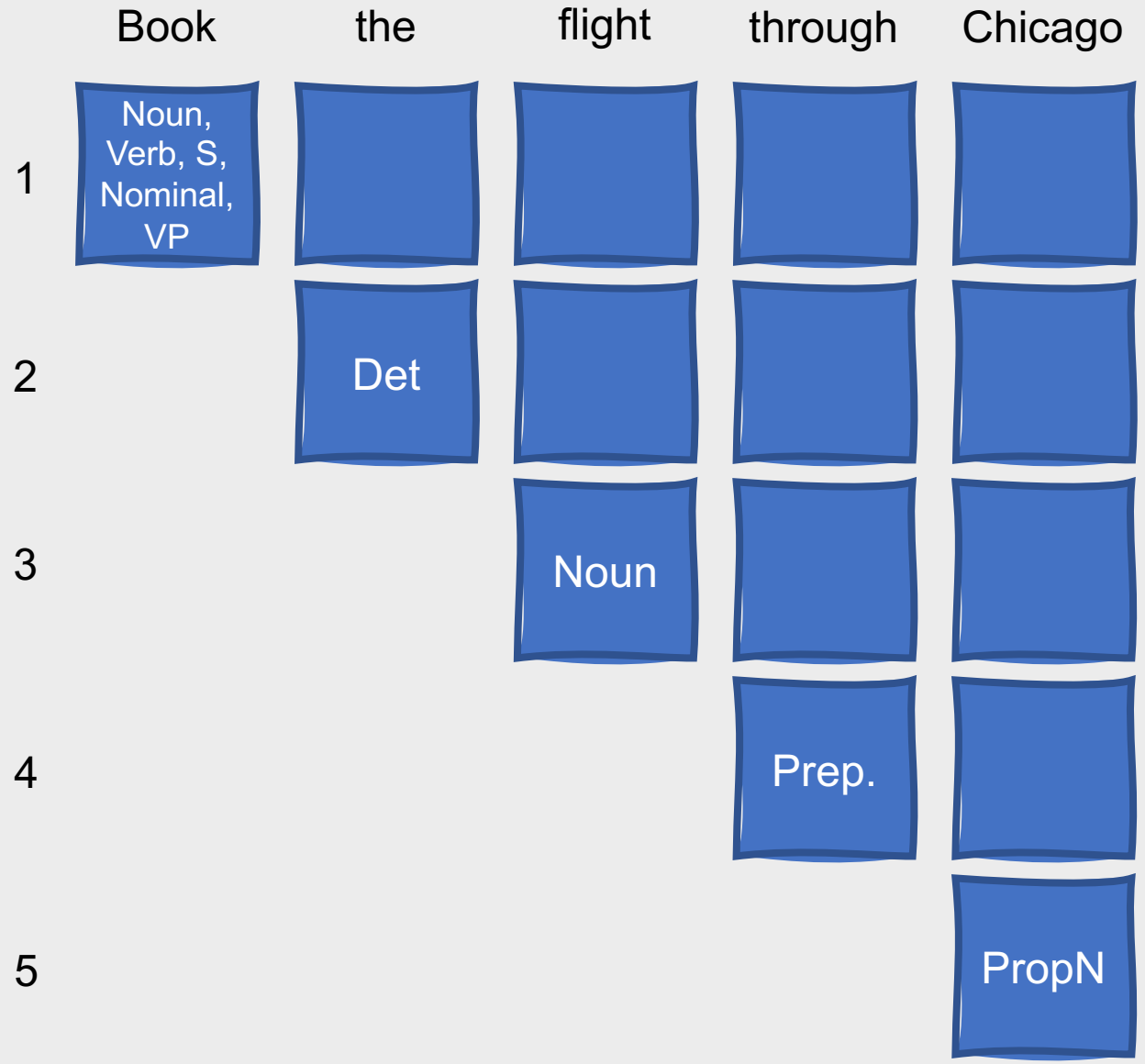
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

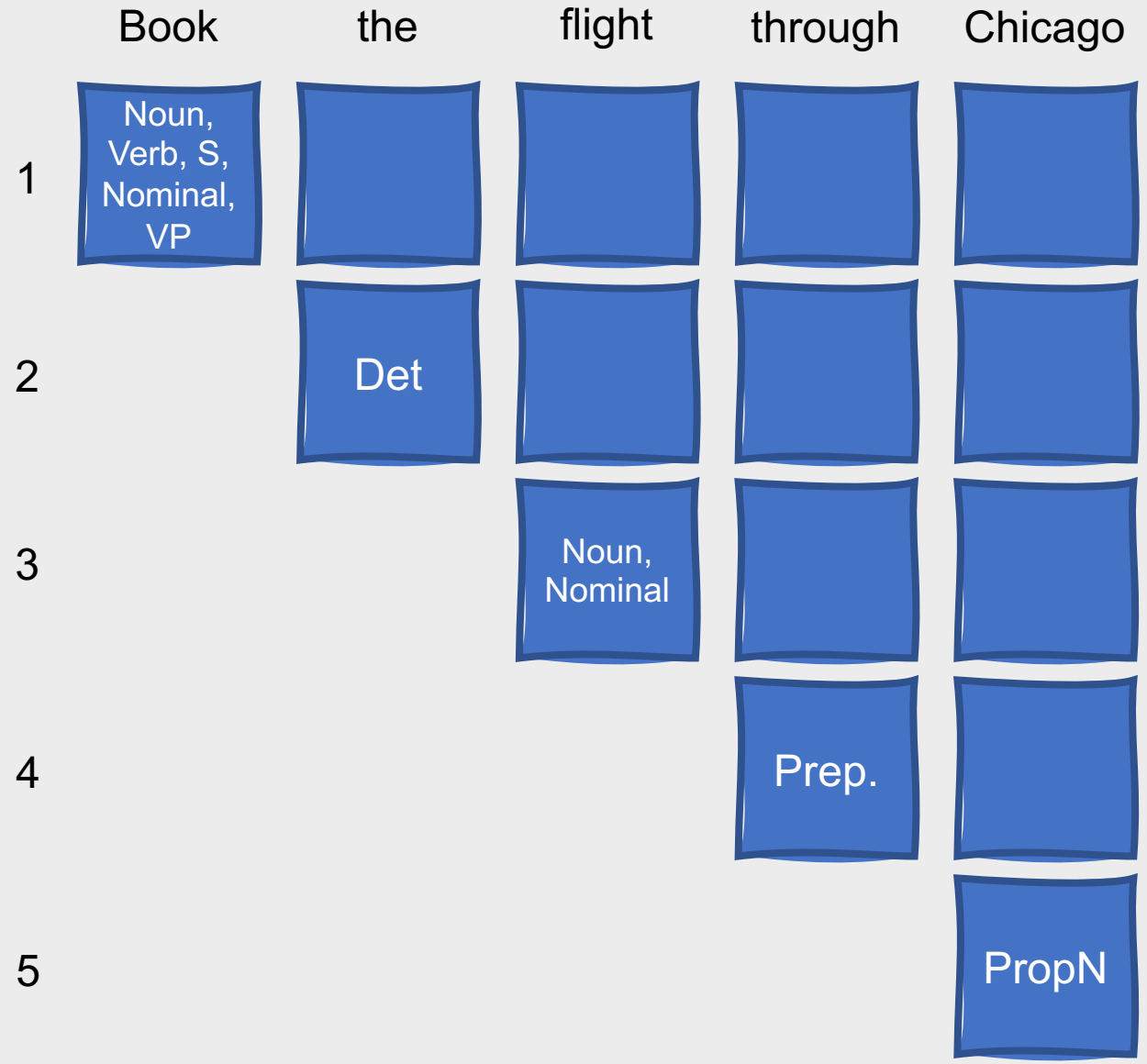
S → NP VP
S → VP → Verb → **book** | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → **book** | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → **book** | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

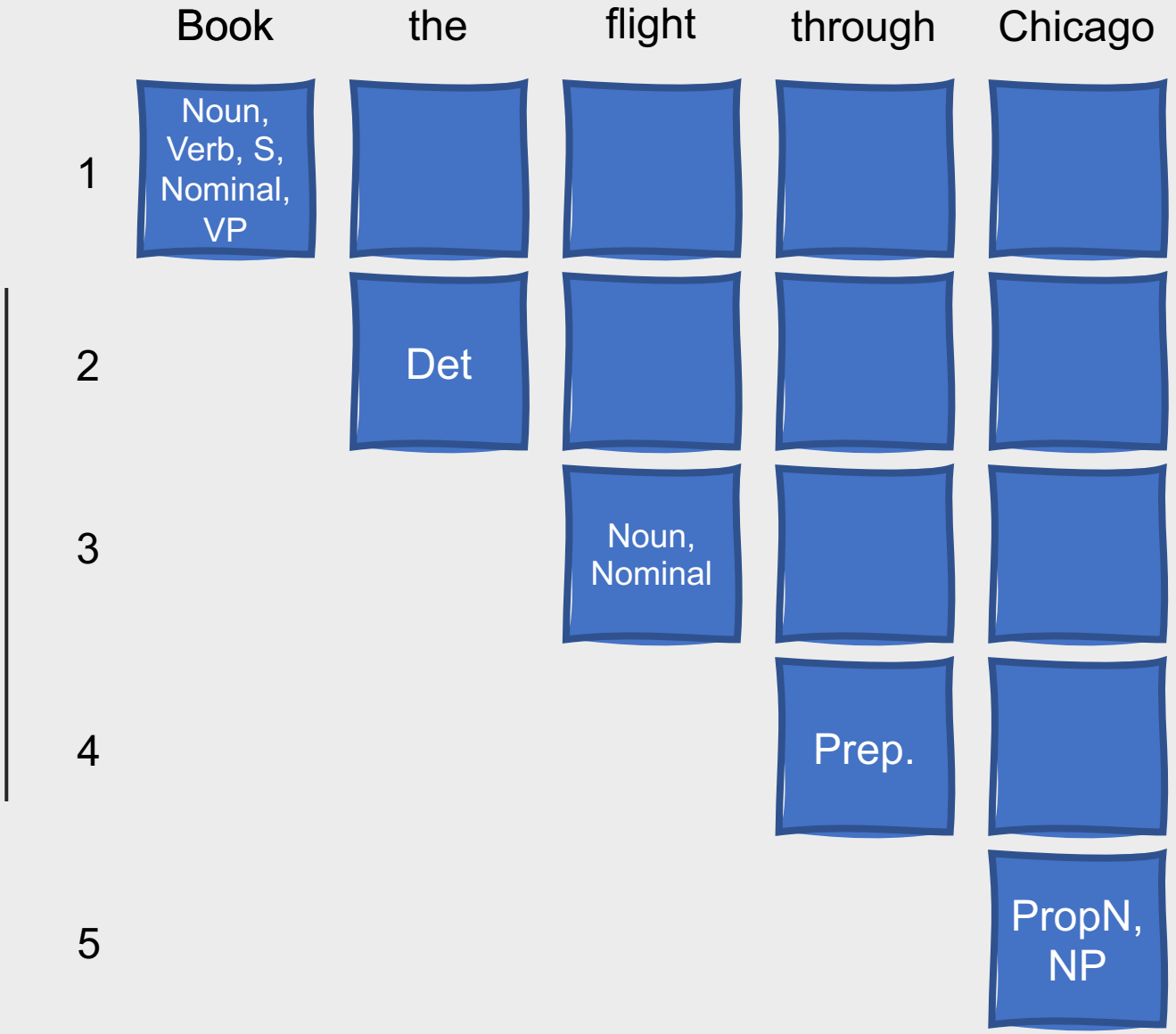
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → **book** | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

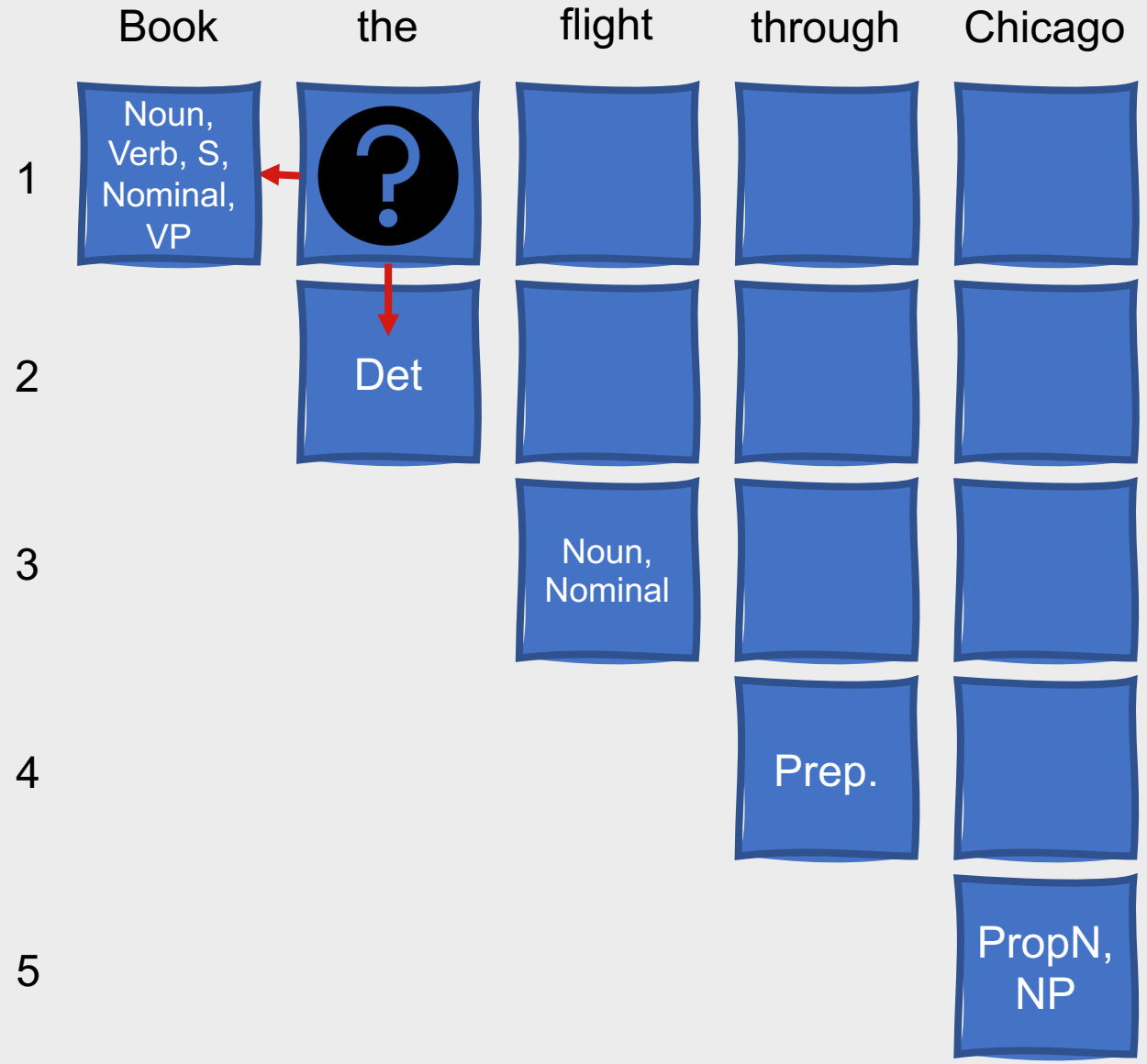
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → **Chicago** | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

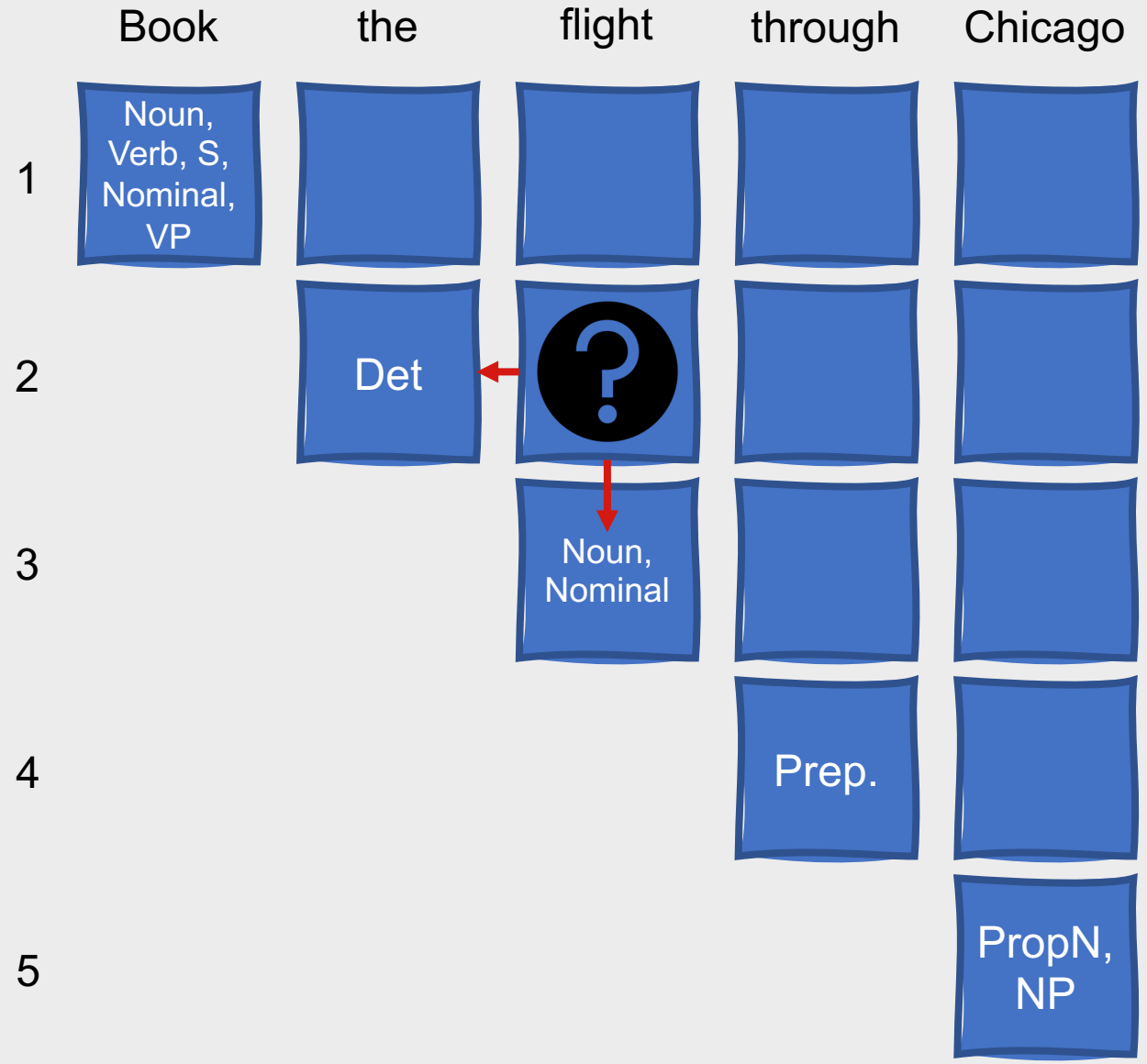
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

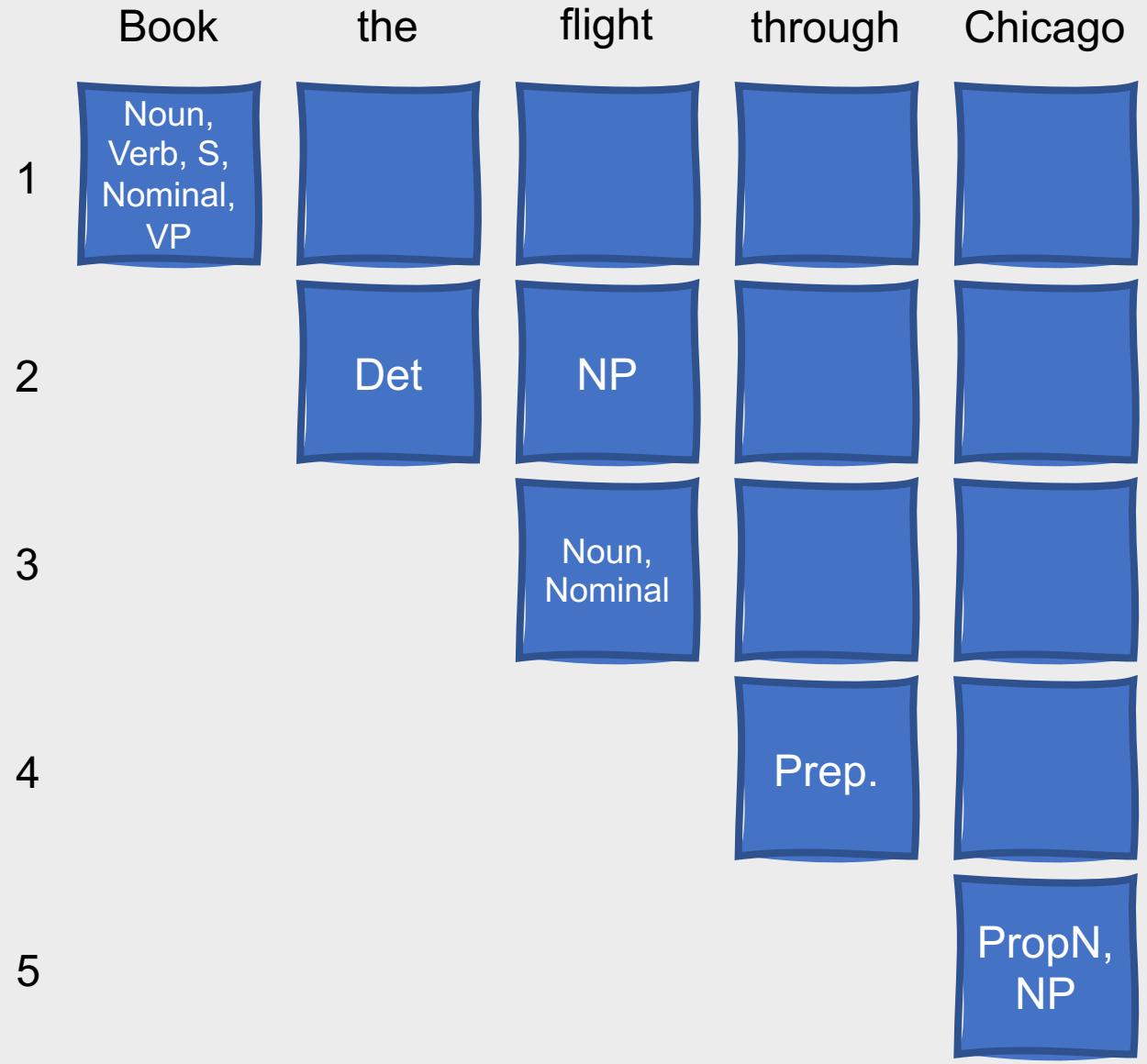
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

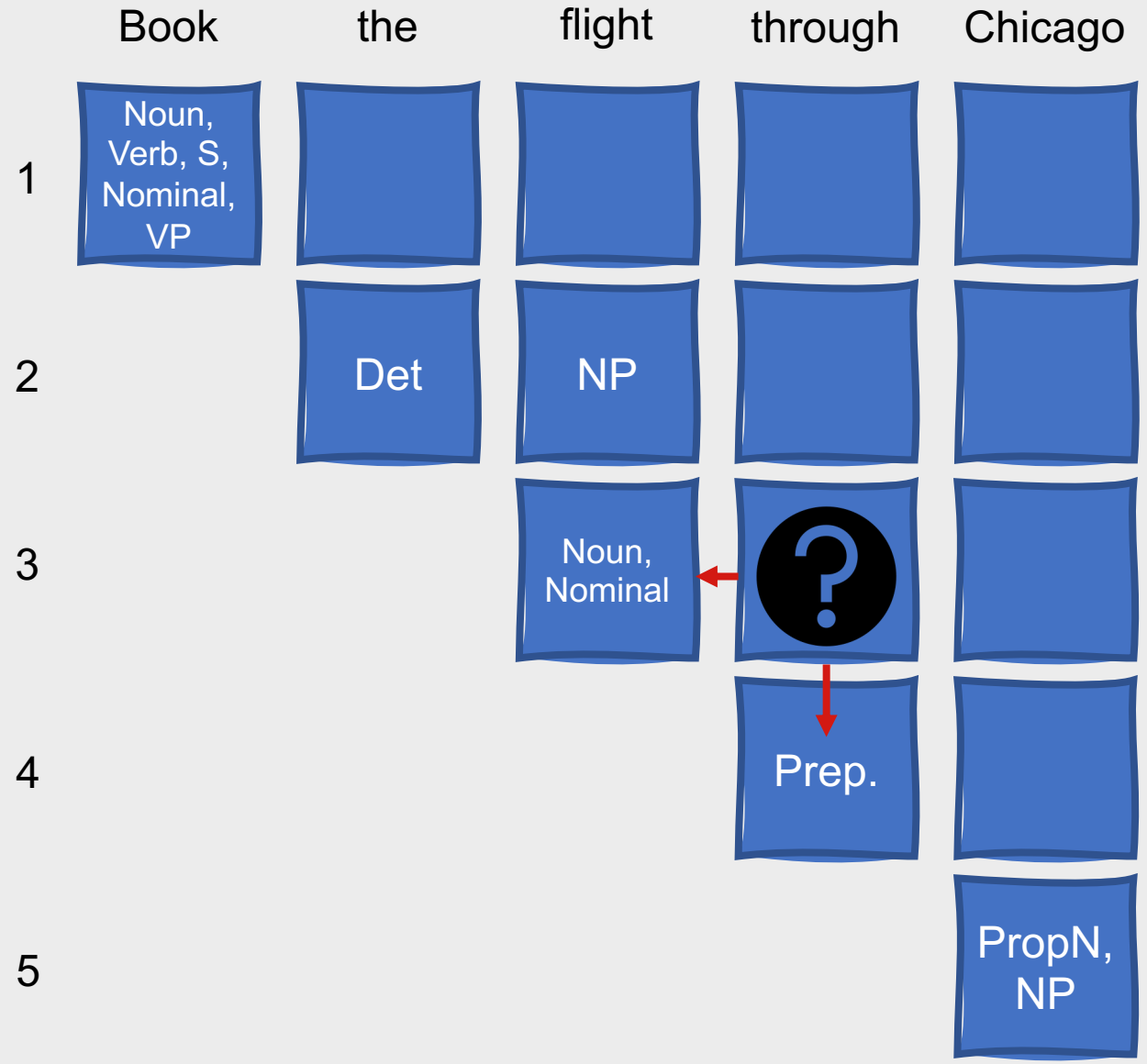
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → **Det Nominal**
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

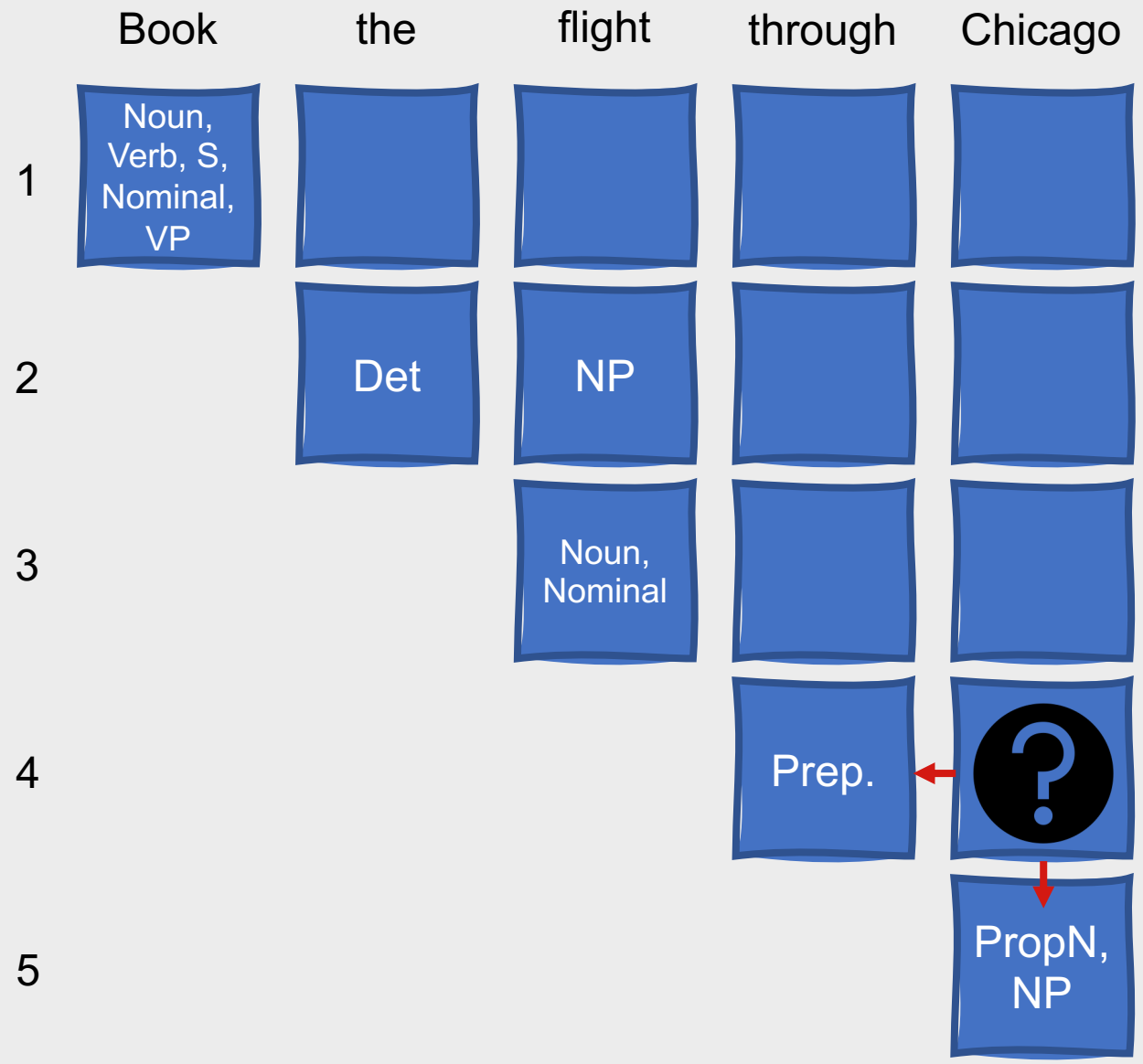
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

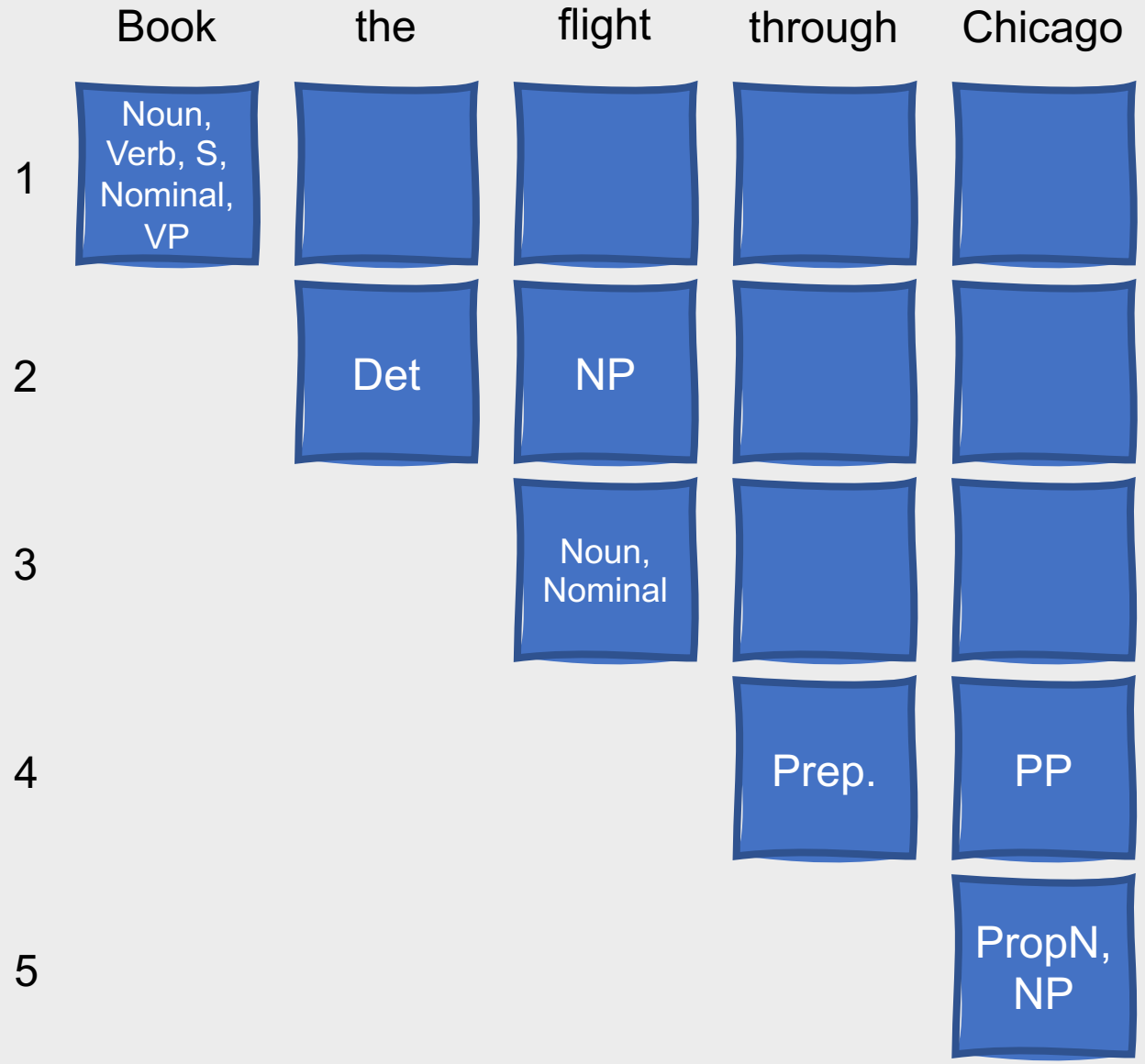
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

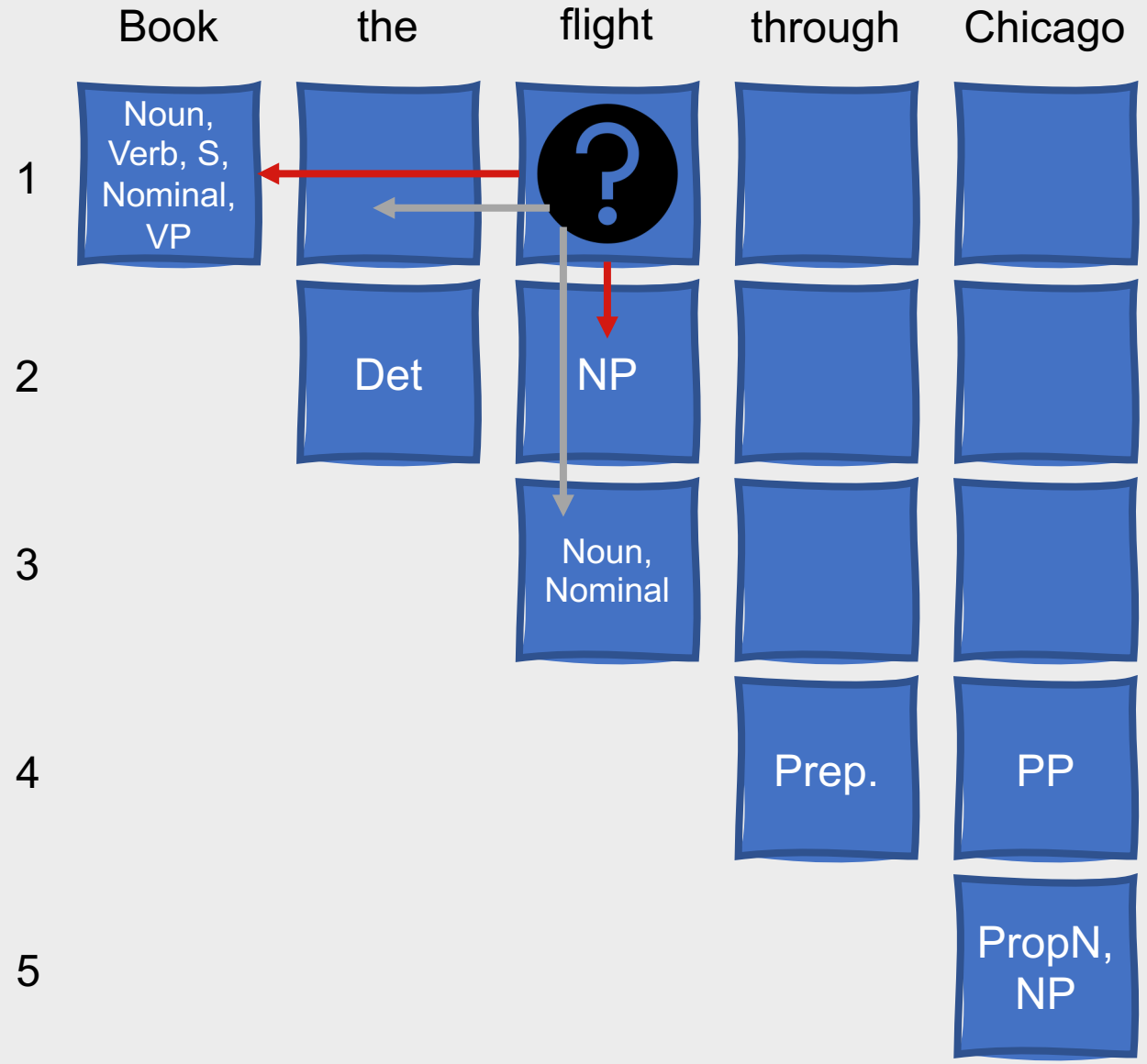
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → **Preposition NP**



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

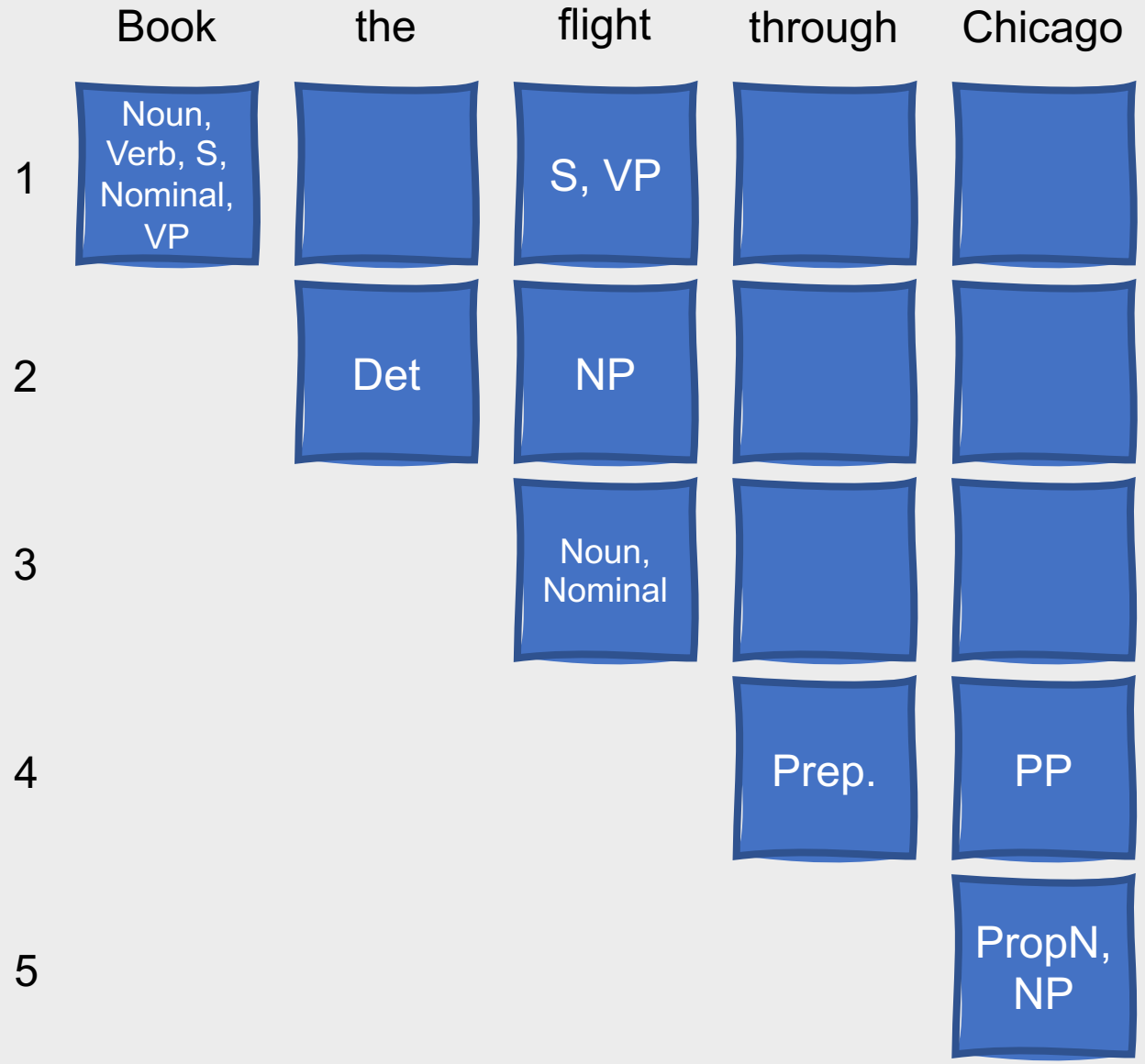
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

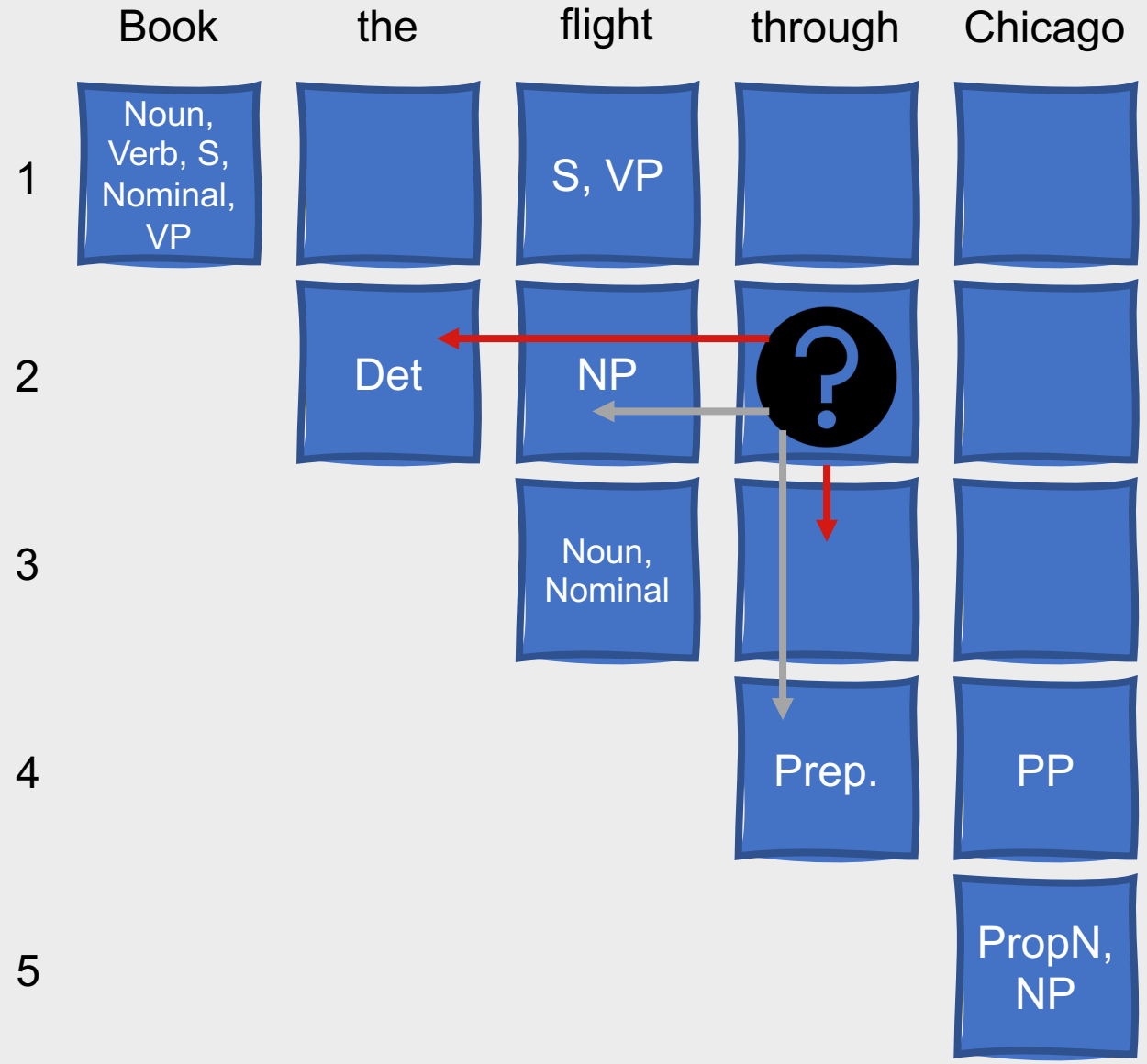
S → NP VP
S → **VP** → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → **Verb NP**
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

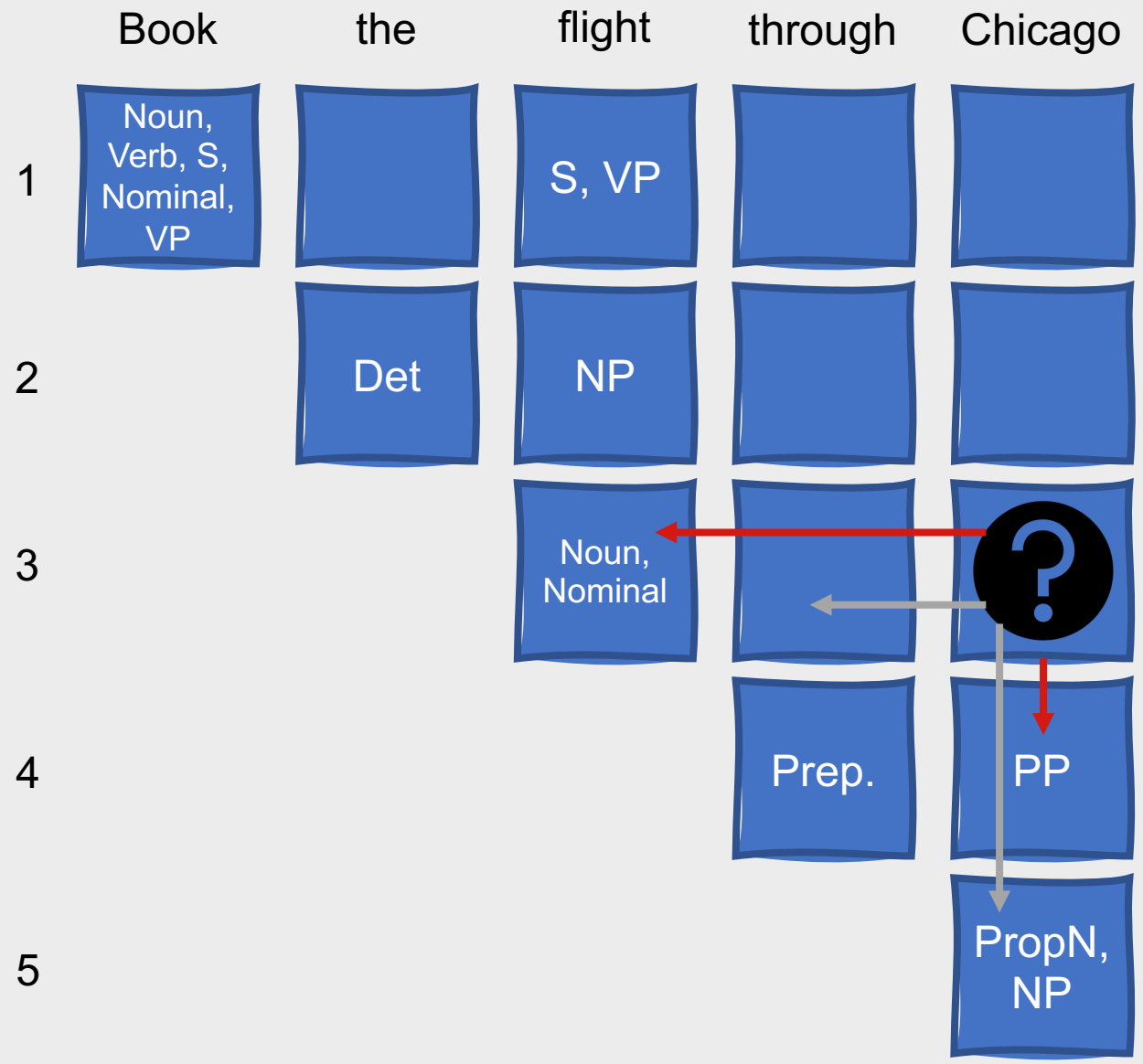
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

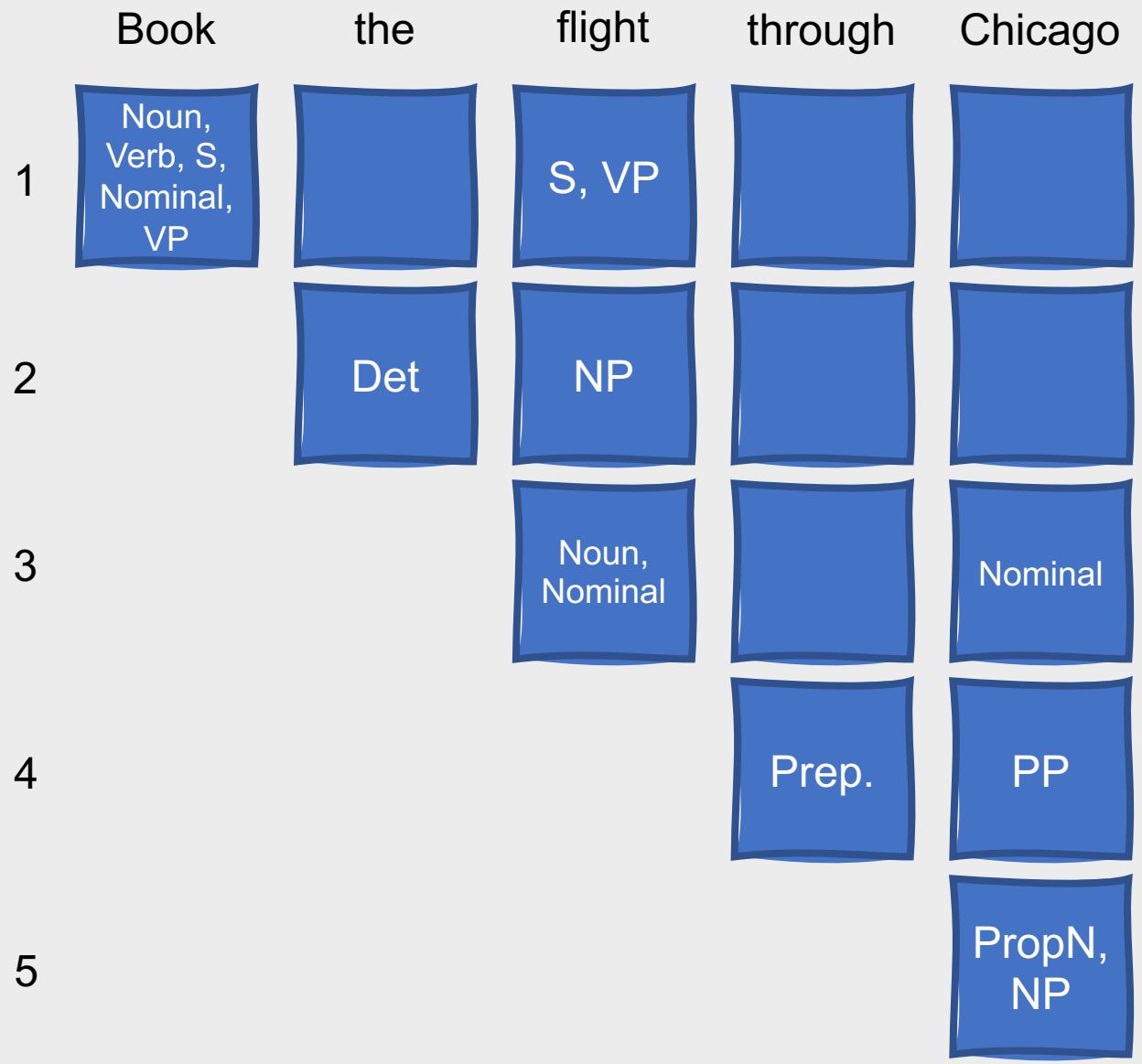
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

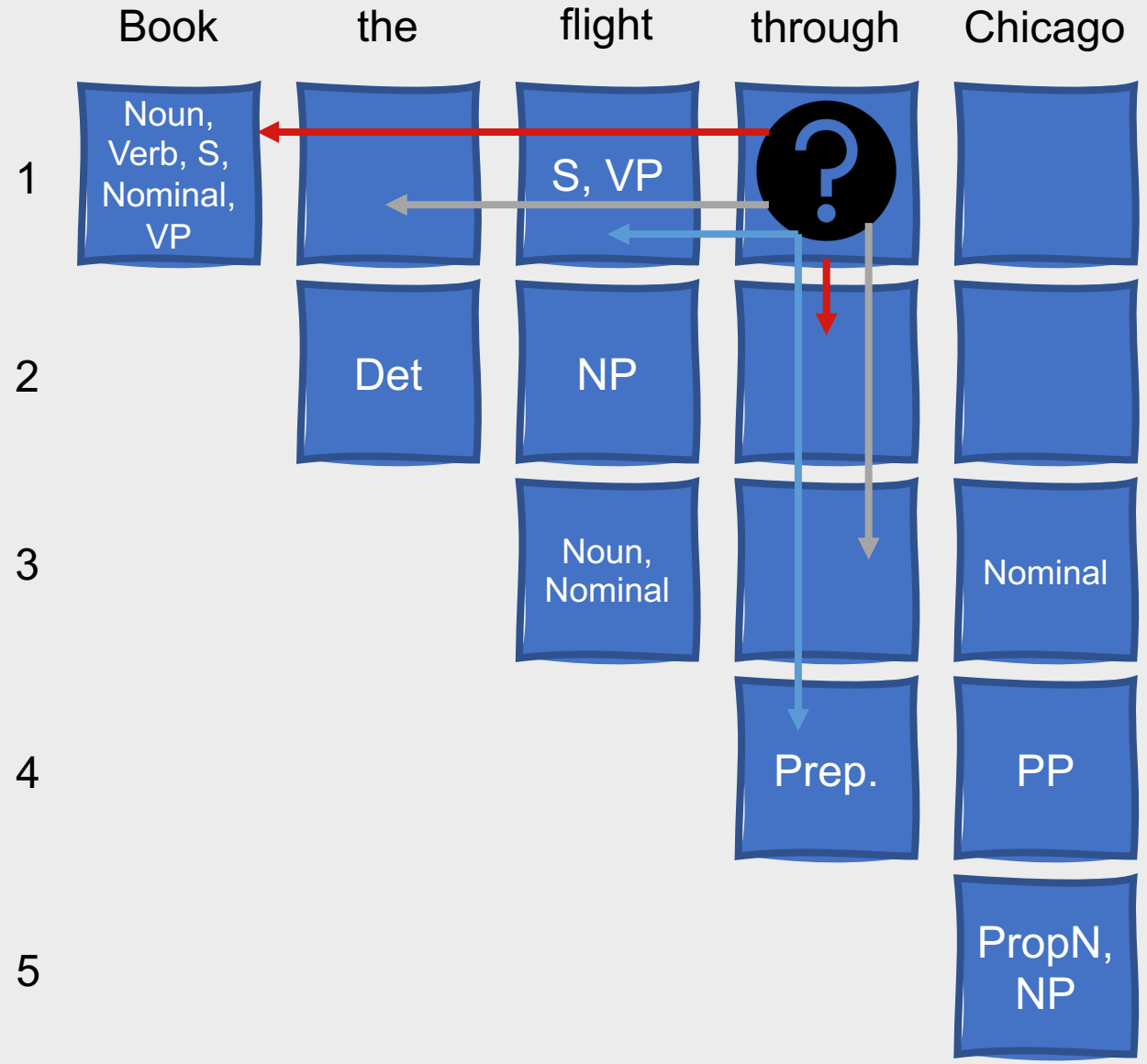
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → **Nominal PP**
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

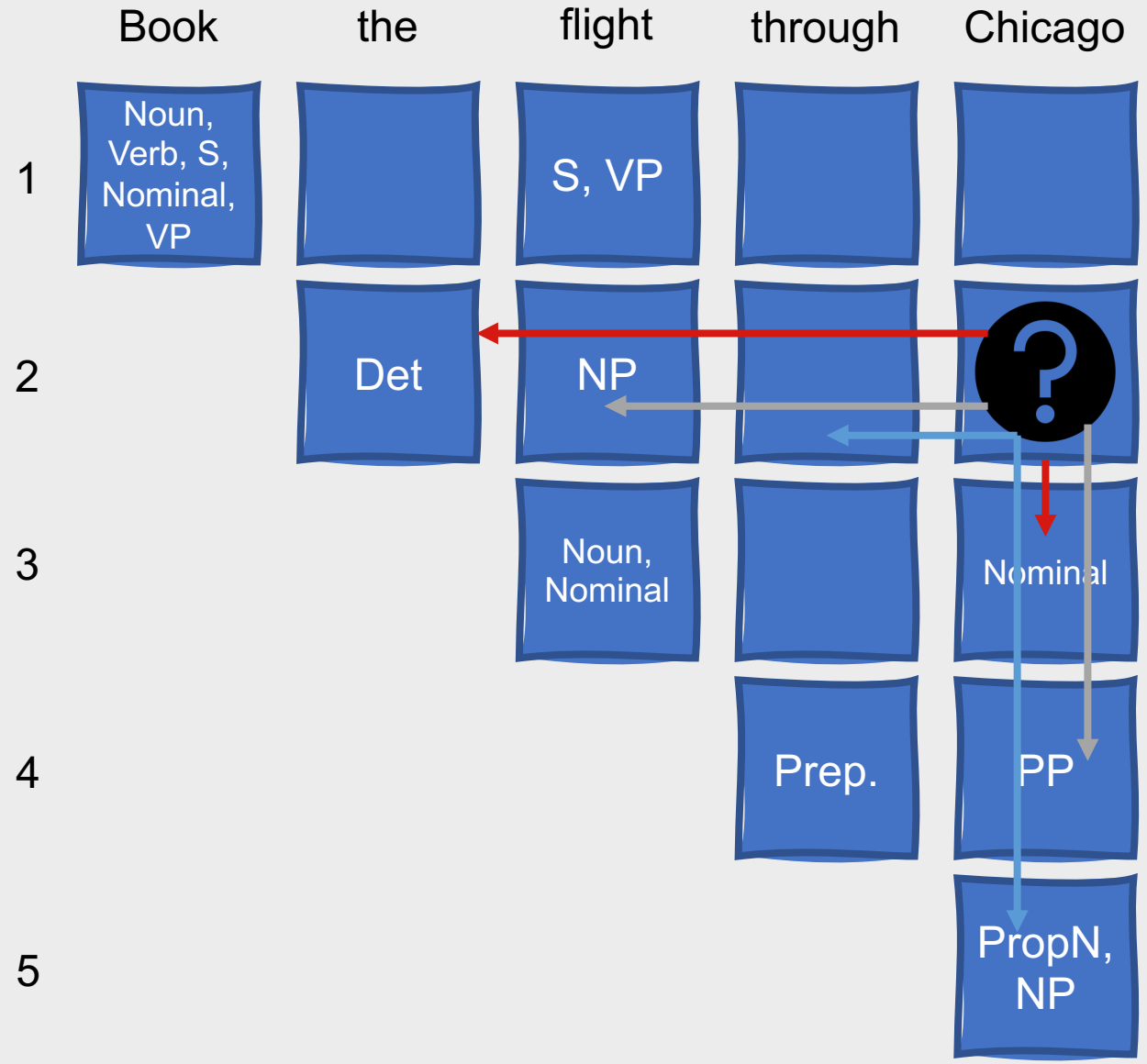
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

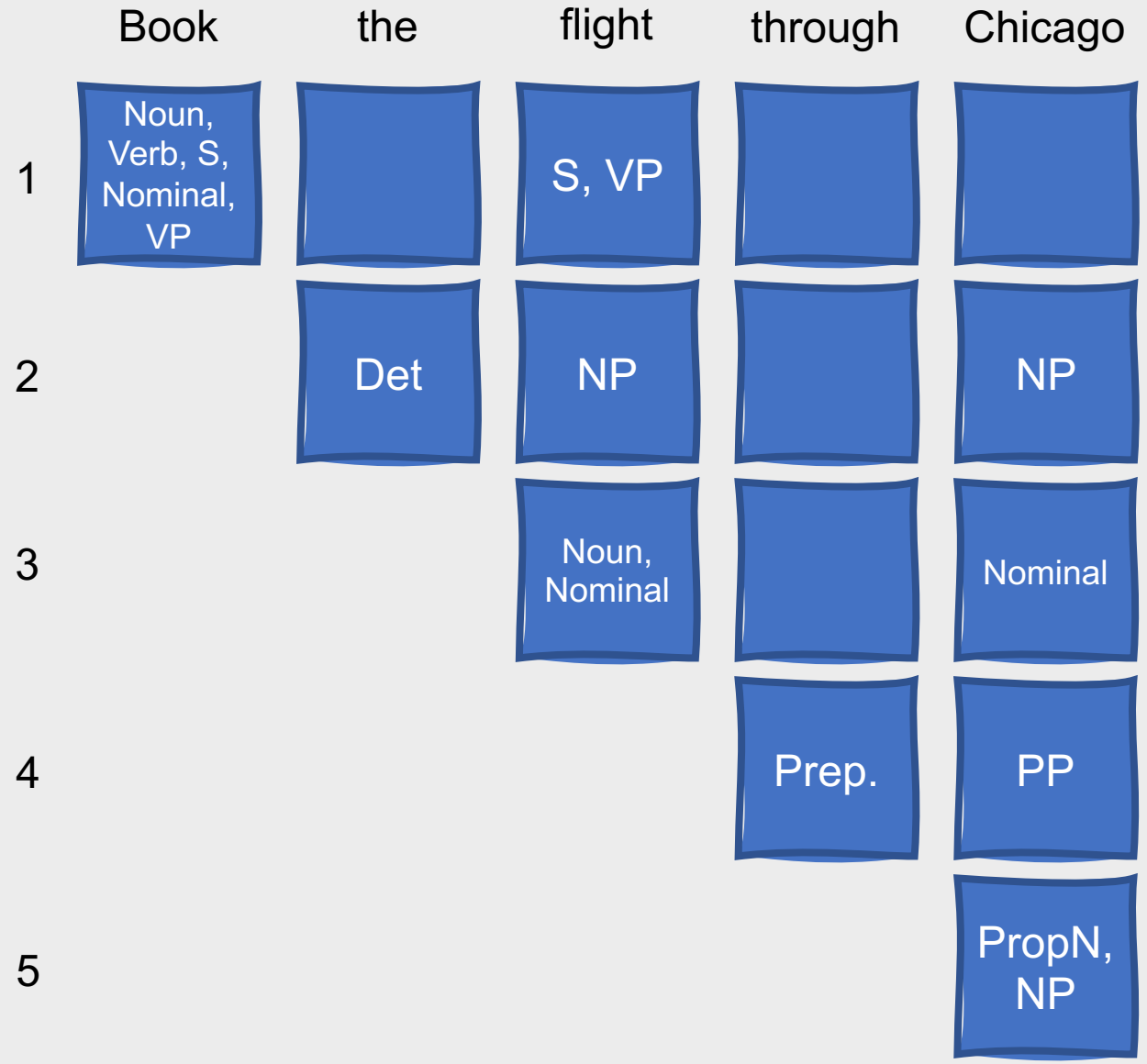
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

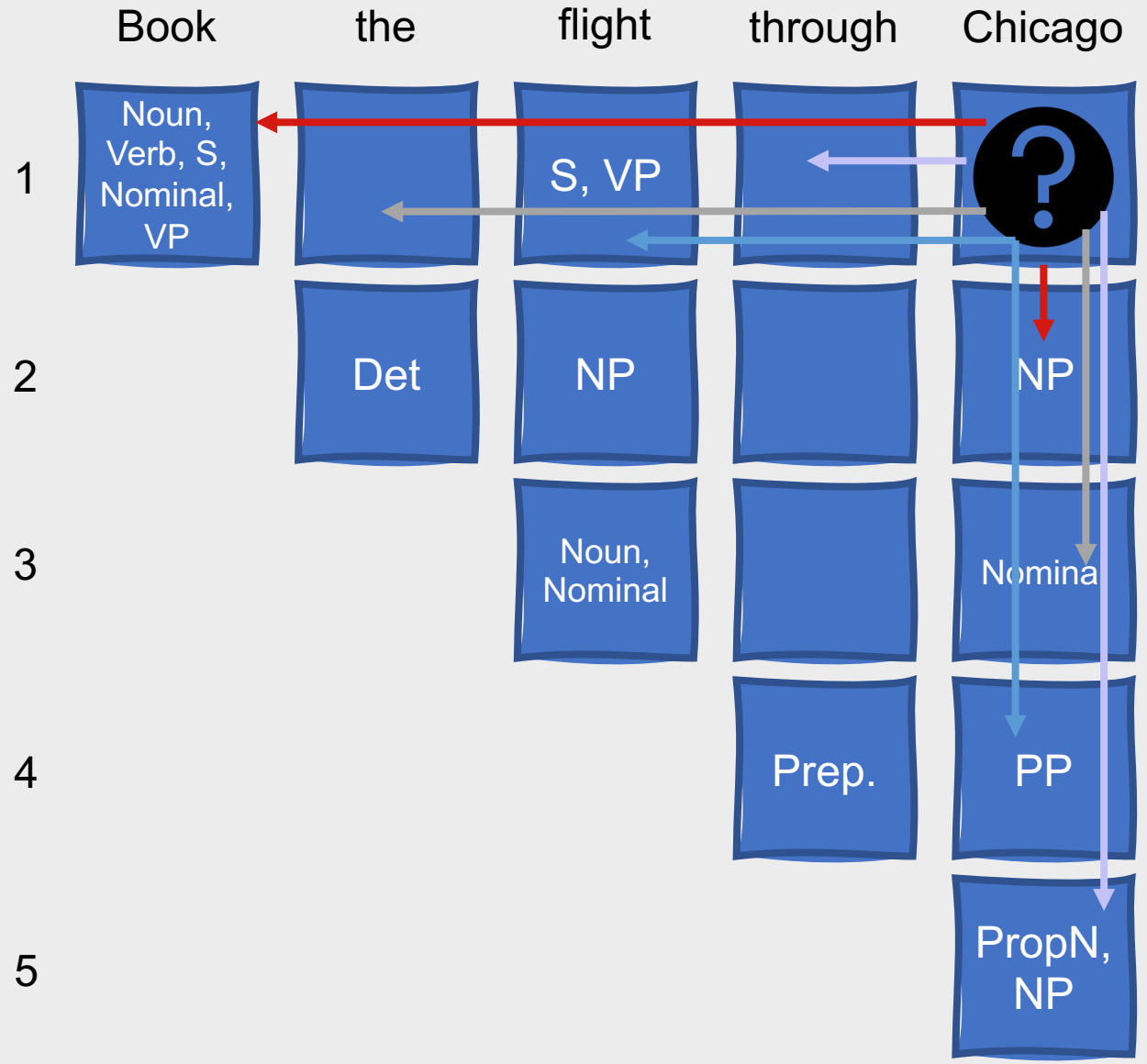
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → **Det Nominal**
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

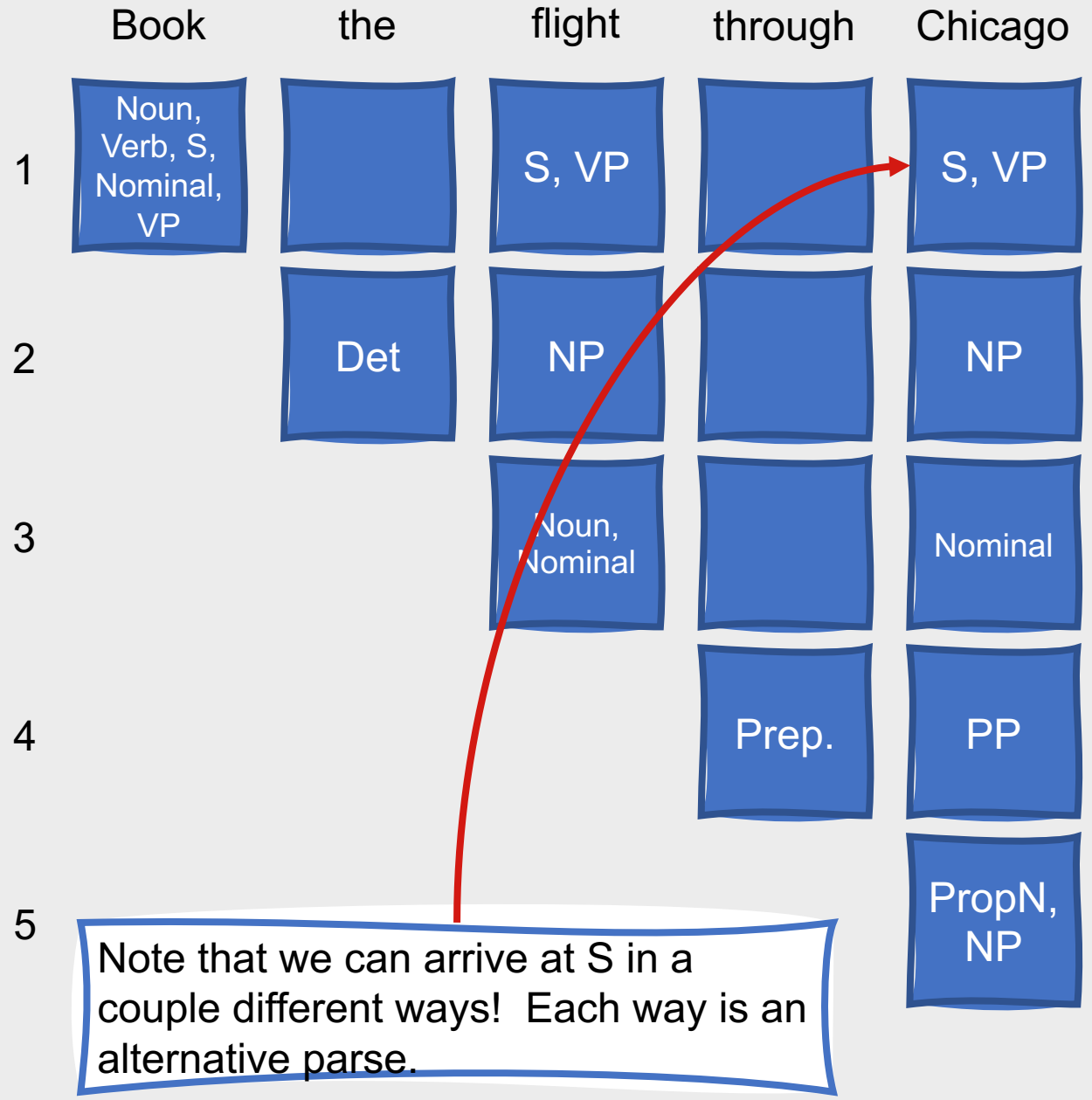
S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

S → NP VP
S → **VP** → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → **Verb NP**
VP → Verb PP
VP → **VP PP**
PP → Preposition NP



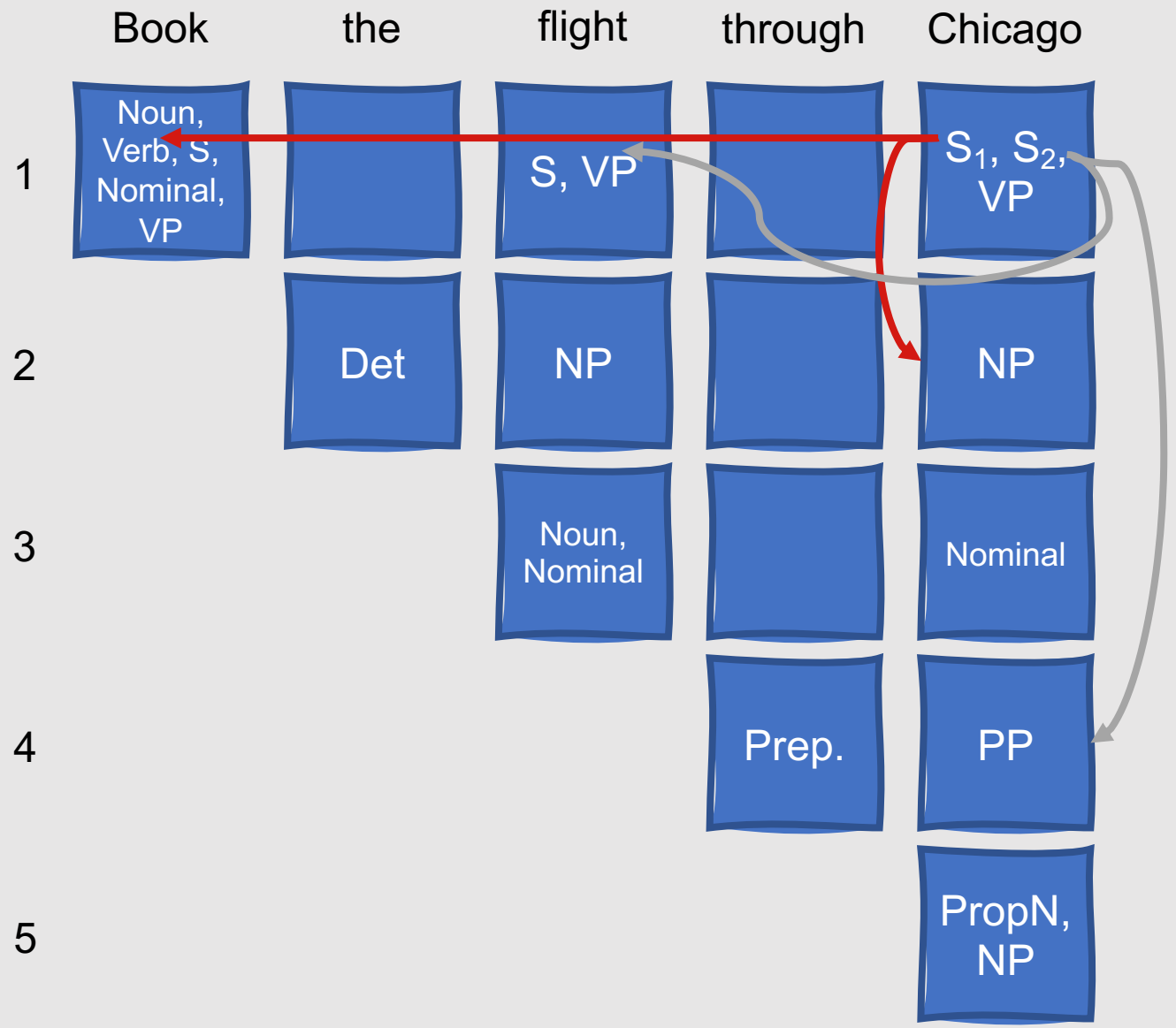
CKY Algorithm

- The example we just saw functions as a **recognizer** …for it to succeed (i.e., find a valid sentence according to this grammar), is simply needs to find an S in cell [0,n]
- To return all possible parses, we need to make two changes to the algorithm:
  - Pair each non-terminal with pointers to the table entries from which it was derived
  - Permit multiple versions of the same non-terminal to be entered into the table
- Then, we can choose an S from cell [0,n] and recursively retrieve its component constituents from the table

CKY Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Chicago | Dallas
Aux → does
Preposition → from | to | on | near | through

S → NP VP
S → VP → Verb → book | include | prefer
NP → Pronoun → I | she | me
NP → Proper-Noun → Chicago | Dallas
NP → Det Nominal
Nominal → Noun → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb → book | include | prefer
VP → Verb NP
VP → Verb PP
VP → VP PP
PP → Preposition NP



CKY Complexity

Time Complexity: $O(n^3)$

Space Complexity: $O(n^2)$

How can we improve upon CKY?

- Fill table in a single sweep over the input words, using a top-down approach
- Table is length *n*+1, where *n* is equivalent to the number of words
- Table entries contain three types of information:
  - A subtree corresponding to a single grammar rule
  - Information about the progress made in completing the subtree
  - The position of the subtree with respect to the input
- This is called Earley parsing

In Earley parsing, table entries are known as states.

- States include structures called **dotted rules**
- A • within the righthand side of a state's grammar rule indicates the progress made towards recognizing it
- A state's **position with respect to the input is represented by two numbers**, indicating (1) where the state begins, and (2) where its dot lies

Example States

- Input: Book that flight.
- S → • VP, [0,0]
  - Top-down prediction for this particular kind of S
  - First 0: Constituent predicted by this state should begin at the start of the input
  - Second 0: Dot lies at the start of the input as well
- NP → Det • Nominal, [1,2]
  - NP begins at position 1
  - Det has been successfully parsed
  - Nominal is expected next
- VP → V NP •, [0,3]
  - Successful discovery of a tree corresponding to a VP that spans the entire input

Earley Algorithm

- An Earley parser moves through the n+1 sets of states in a chart in a left-to-right fashion, processing the states within each set in order
- At each step, one of three operators is applied to each state depending on its status
  - Predictor
  - Scanner
  - Completer
- This results in the addition of new states to the end of either the current or next set of states in the chart
- States are never removed
- The algorithm never backtracks
- The presence of S $\rightarrow \alpha$ •, [0,n] in the list of states in the last chart entry indicates a successful parse

Earley Operators: Predictor

Predictor

- Creates new states representing top-down expectations
- Applied to any state that has a non-terminal immediately to the right of its dot (as long as the non-terminal is not a POS category)
- New states are placed into the same chart entry as the generating state
- They begin and end at the same point in the input where the generating state ends

S → • VP, [0,0]

- VP → • Verb, [0,0]
- VP → • Verb NP, [0,0]
- VP → • Verb NP PP, [0,0]
- VP → • Verb PP, [0,0]
- VP → • VP PP, [0,0]

Earley Operators: Scanner

- Used when a state has a POS category to the right of the dot
- Examines input and incorporates a state corresponding to the prediction of a word with a particular POS into the chart
- Does so by creating a new state from the input state with the dot advanced over the predicted input category
- VP → • Verb NP, [0,0]
  - Since category following the dot is a part of speech (Verb)….
  - Verb → book •, [0,1]

Earley Operators: Completer

- Applied to a state when its dot has reached the right end of the rule
- Indicates that the parser has successfully discovered a particular grammatical category over some span of input
- Purpose: Find and advance all previously created states that were looking for this grammatical category at this position in the input
- New states are created by copying the older state, advancing the dot over the expected category, and installing the new state in the current chart entry
- NP → Det Nominal •, [1,3]
  - What incomplete states end at position 1 and expect an NP?
  - VP → Verb • NP, [0,1]
  - VP → Verb • NP PP, [0,1]
  - So, add VP → Verb NP •, [0,3] and the new incomplete VP → Verb NP • PP, [0,3] to the chart

Earley Algorithm: Example

Chart	State	Rule	Start, End	Operator
0	S0	$\gamma \rightarrow \bullet$ S	0, 0	Start State
0	S1	S $\rightarrow \bullet$ NP VP	0, 0	Predictor
0	S2	S $\rightarrow \bullet$ VP	0, 0	Predictor
0	S3	NP $\rightarrow \bullet$ Det Nominal	0, 0	Predictor
0	S4	VP $\rightarrow \bullet$ Verb	0, 0	Predictor
0	S5	VP $\rightarrow \bullet$ Verb NP	0, 0	Predictor

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer

S → NP VP
S → VP
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
VP → Verb
VP → Verb NP

Earley Algorithm: Example

Chart	State	Rule	Start, End	Operator
0	S0	$\gamma \rightarrow \bullet$ S	0, 0	Start State
0	S1	S $\rightarrow \bullet$ NP VP	0, 0	Predictor
0	S2	S $\rightarrow \bullet$ VP	0, 0	Predictor
0	S3	NP $\rightarrow \bullet$ Det Nominal	0, 0	Predictor
0	S4	VP $\rightarrow \bullet$ Verb	0, 0	Predictor
0	S5	VP $\rightarrow \bullet$ Verb NP	0, 0	Predictor
1	S6	Verb $\rightarrow$ book $\bullet$	0, 1	Scanner
1	S7	VP $\rightarrow$ Verb $\bullet$	0, 1	Completer
1	S8	VP $\rightarrow$ Verb $\bullet$ NP	0, 1	Completer
1	S9	S $\rightarrow$ VP $\bullet$	0, 1	Completer
1	S10	NP $\rightarrow \bullet$ Det Nominal	1, 1	Predictor

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer

S → NP VP
S → VP
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
VP → Verb
VP → Verb NP

Earley Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer

S → NP VP
S → VP
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
VP → Verb
VP → Verb NP

Chart	State	Rule	Start, End	Operator
0	S0	γ → • S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor
0	S2	S → • VP	0, 0	Predictor
0	S3	NP → • Det Nominal	0, 0	Predictor
0	S4	VP → • Verb	0, 0	Predictor
0	S5	VP → • Verb NP	0, 0	Predictor
1	S6	Verb → book •	0, 1	Scanner
1	S7	VP → Verb •	0, 1	Completer
1	S8	VP → Verb • NP	0, 1	Completer
1	S9	S → VP •	0, 1	Completer
1	S10	NP → • Det Nominal	1, 1	Predictor
2	S11	Det → that •	1, 2	Scanner
2	S12	NP → Det • Nominal	1, 2	Completer
2	S13	Nominal → • Noun	2, 2	Predictor
2	S14	Nominal → • Nominal Noun	2, 2	Predictor

Earley Algorithm: Example

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer

S → NP VP
S → VP
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
VP → Verb
VP → Verb NP

Chart	State	Rule	Start, End	Operator
0	S0	γ → • S	0, 0	Start State
0	S1	S → • NP VP	0, 0	Predictor
0	S2	S → • VP	0, 0	Predictor
0	S3	NP → • Det Nominal	0, 0	Predictor
0	S4	VP → • Verb	0, 0	Predictor
0	S5	VP → • Verb NP	0, 0	Predictor
1	S6	Verb → book •	0, 1	Scanner
1	S7	VP → Verb •	0, 1	Completer
1	S8	VP → Verb • NP	0, 1	Completer
1	S9	S → VP •	0, 1	Completer
1	S10	NP → • Det Nominal	1, 1	Predictor
2	S11	Det → that •	1, 2	Scanner
2	S12	NP → Det • Nominal	1, 2	Completer
2	S13	Nominal → • Noun	2, 2	Predictor
2	S14	Nominal → • Nominal Noun	2, 2	Predictor
3	S15	Noun → flight •	2, 3	Scanner
3	S16	Nominal → Noun •	2, 3	Completer
3	S17	NP → Det Nominal •	1, 3	Completer
3	S18	Nominal → Nominal • Noun	2, 3	Completer
3	S19	VP → Verb NP •	0, 3	Completer
3	S20	S → VP •	0, 3	Completer

Which states participate in the final parse?

Chart	State	Rule	Start, End	Operator
0	S0	$\gamma \rightarrow \bullet$ S	0, 0	Start State
0	S1	S $\rightarrow \bullet$ NP VP	0, 0	Predictor
0	S2	S $\rightarrow \bullet$ VP	0, 0	Predictor
0	S3	NP $\rightarrow \bullet$ Det Nominal	0, 0	Predictor
0	S4	VP $\rightarrow \bullet$ Verb	0, 0	Predictor
0	S5	VP $\rightarrow \bullet$ Verb NP	0, 0	Predictor
1	S6	Verb $\rightarrow$ book $\bullet$	0, 1	Scanner
1	S7	VP $\rightarrow$ Verb $\bullet$	0, 1	Completer
1	S8	VP $\rightarrow$ Verb $\bullet$ NP	0, 1	Completer
1	S9	S $\rightarrow$ VP $\bullet$	0, 1	Completer
1	S10	NP $\rightarrow \bullet$ Det Nominal	1, 1	Predictor
2	S11	Det $\rightarrow$ that $\bullet$	1, 2	Scanner
2	S12	NP $\rightarrow$ Det $\bullet$ Nominal	1, 2	Completer
2	S13	Nominal $\rightarrow \bullet$ Noun	2, 2	Predictor
2	S14	Nominal $\rightarrow \bullet$ Nominal Noun	2, 2	Predictor
3	S15	Noun $\rightarrow$ flight $\bullet$	2, 3	Scanner
3	S16	Nominal $\rightarrow$ Noun $\bullet$	2, 3	Completer
3	S17	NP $\rightarrow$ Det Nominal $\bullet$	1, 3	Completer
3	S18	Nominal $\rightarrow$ Nominal $\bullet$ Noun	2, 3	Completer
3	S19	VP $\rightarrow$ Verb NP $\bullet$	0, 3	Completer
3	S20	S $\rightarrow$ VP $\bullet$	0, 3	Completer

As with CKY, the example algorithm acted as a recognizer.

- We can retrieve parse trees by adding a field to store information about the completed states that generated constituents
- How to do this?
  - Have the Completer operator add a pointer to the previous state onto a list of constituent states for the new state
  - When an S is found in the final chart, just follow pointers backward

Which states participate in the final parse?

Chart	State	Rule	Start, End	Operator (Backward Pointer)
0	S0	$\gamma \rightarrow \bullet$ S	0, 0	Start State
0	S1	S $\rightarrow \bullet$ NP VP	0, 0	Predictor
0	S2	S $\rightarrow \bullet$ VP	0, 0	Predictor
0	S3	NP $\rightarrow \bullet$ Det Nominal	0, 0	Predictor
0	S4	VP $\rightarrow \bullet$ Verb	0, 0	Predictor
0	S5	VP $\rightarrow \bullet$ Verb NP	0, 0	Predictor
1	S6	Verb $\rightarrow$ book $\bullet$	0, 1	Scanner
1	S7	VP $\rightarrow$ Verb $\bullet$	0, 1	Completer
1	S8	VP $\rightarrow$ Verb $\bullet$ NP	0, 1	Completer
1	S9	S $\rightarrow$ VP $\bullet$	0, 1	Completer
1	S10	NP $\rightarrow \bullet$ Det Nominal	1, 1	Predictor
2	S11	Det $\rightarrow$ that $\bullet$	1, 2	Scanner
2	S12	NP $\rightarrow$ Det $\bullet$ Nominal	1, 2	Completer
2	S13	Nominal $\rightarrow \bullet$ Noun	2, 2	Predictor
2	S14	Nominal $\rightarrow \bullet$ Nominal Noun	2, 2	Predictor
3	S15	Noun $\rightarrow$ flight $\bullet$	2, 3	Scanner
3	S16	Nominal $\rightarrow$ Noun $\bullet$	2, 3	Completer (S15)
3	S17	NP $\rightarrow$ Det Nominal $\bullet$	1, 3	Completer (S11, S15)
3	S18	Nominal $\rightarrow$ Nominal $\bullet$ Noun	2, 3	Completer
3	S19	VP $\rightarrow$ Verb NP $\bullet$	0, 3	Completer (S6, S17)
3	S20	S $\rightarrow$ VP $\bullet$	0, 3	Completer (S19)

Summary: Syntactic Parsing

- **Syntactic parsing** is the process of automatically determining the grammatical structure of an input sentence
- Language is ambiguous, so **sentences can have multiple grammatically-correct parses**
- Parsing can be performed using either a **top-down** or **bottom-up** approach
- Common algorithms for syntactic parsing include:
  - **CKY**
  - **Earley**

What is dependency parsing?

- Automatically determining **directed grammatical (and semantic!) relationships** between words in a sentence.
  - **Syntactic**: Focused on **sentence structure**
  - **Semantic**: Focused on **meaning**

How are dependency grammars different from CFGs?

- CFGs are used to automatically generate constituent-based representations
  - Noun phrases, verb phrases, etc.
- Dependency grammars ignore phrase-structure rules, and instead define sentence structure in terms of the relationships between individual words
  - Nominal subject, direct object, etc.
- For both, labels are still drawn from a fixed inventory of grammatical relations

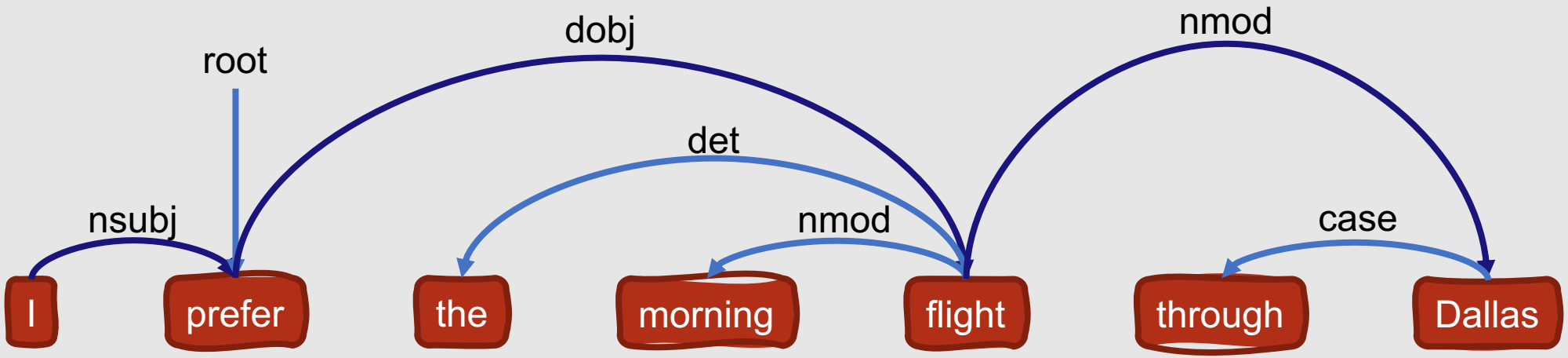
Dependency grammars can deal with languages that are morphologically rich and have a relatively free word order.

Morphologically rich: More inflections (changes to words that influence meaning or grammatical relation)

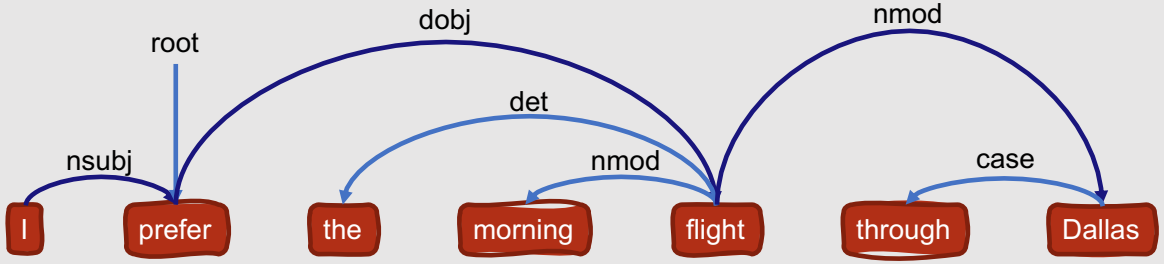
Free word order: Words can be moved around in a sentence but the overall meaning will remain the same (syntax is less important)

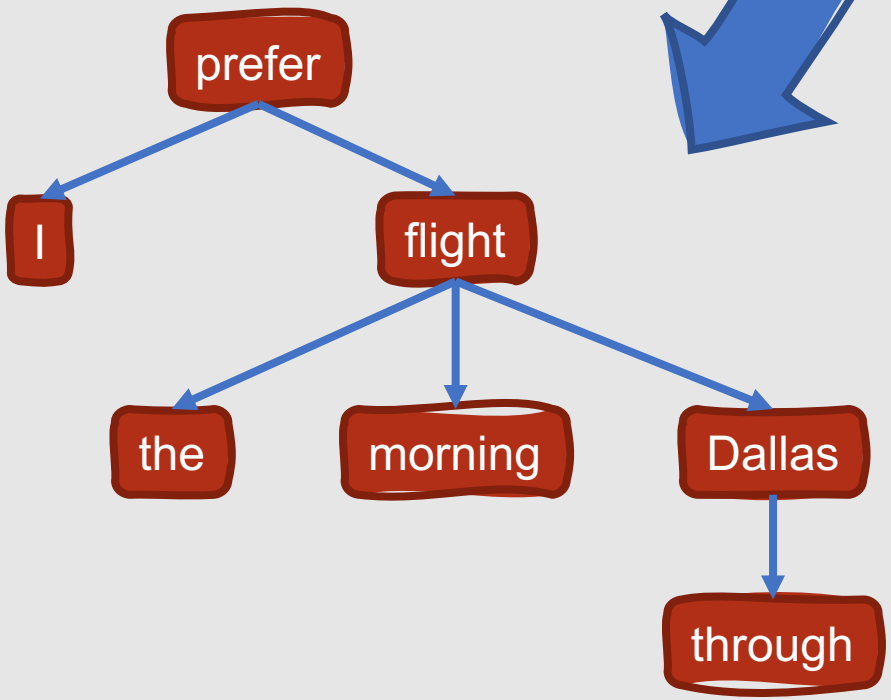
Typically, there is a trade-off between morphological richness and importance of syntax

Typed Dependency Structure

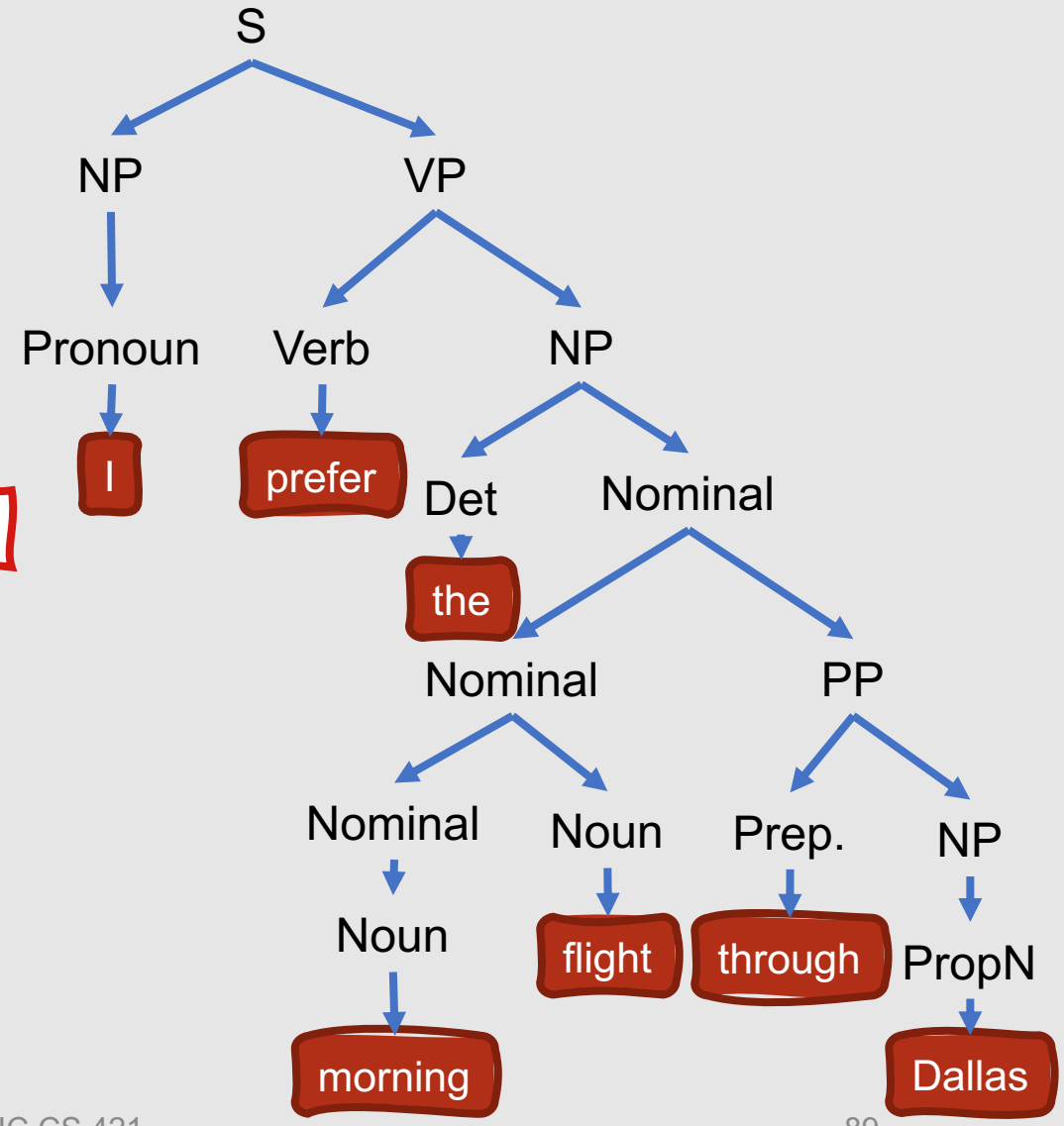


Comparison with Syntactic Parse





vs.



Why is dependency parsing useful?

- Dependency parsing provides an approximation of the semantic relationships between different words in a sentence and their arguments
- This information is useful for many NLP applications, including:
  - Coreference resolution
  - Question answering
  - Information extraction

Dependency Relations

- Two components:
  - **Head**
  - **Dependent**
- **Heads** are linked to the words that are immediately **dependent** on them
- Relation types describe the **dependent**'s role with respect to its **head**
  - Subject
  - Direct object
  - Indirect object

Dependency Relations

- Relation types tend to correlate with sentence position and constituent type in English, but there is not an explicit connection between these elements
- In more flexible languages (e.g., those with relatively free word order), the information encoded in these relation types often cannot be estimated otherwise (e.g., by using a constituent tree)

Just like with CFGs, there are a variety of taxonomies that can be used to label dependencies between words.

These are often referred to as dependency treebanks

A few of the most popular dependency treebanks include:

- Stanford dependencies
- CoNLL dependencies
- Universal dependencies

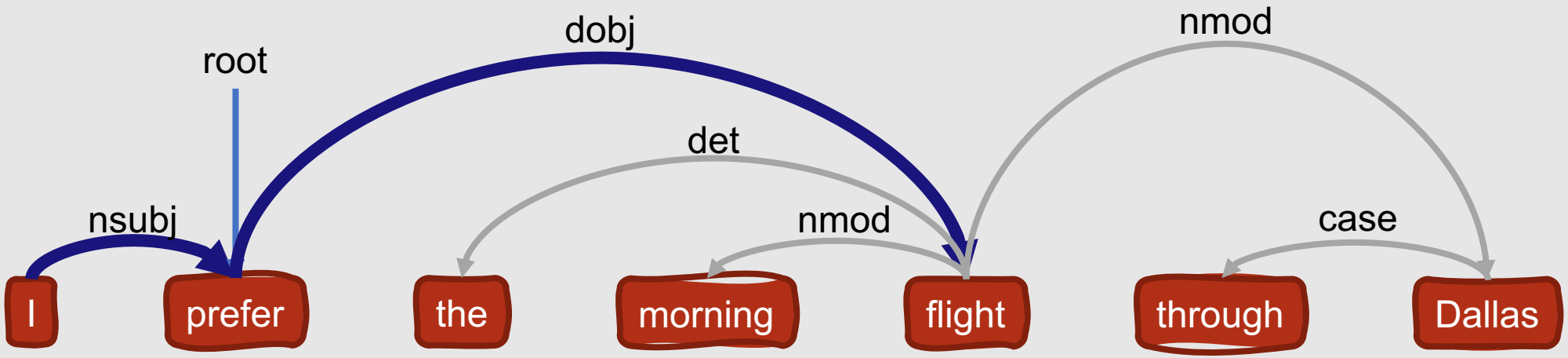
Just like with other corpora we've discussed so far, these treebanks are typically created by either:

- Having human annotators manually create dependency structures for a collection of sentences
- Automatically creating initial dependency structures and then having human annotators manually correct those structures

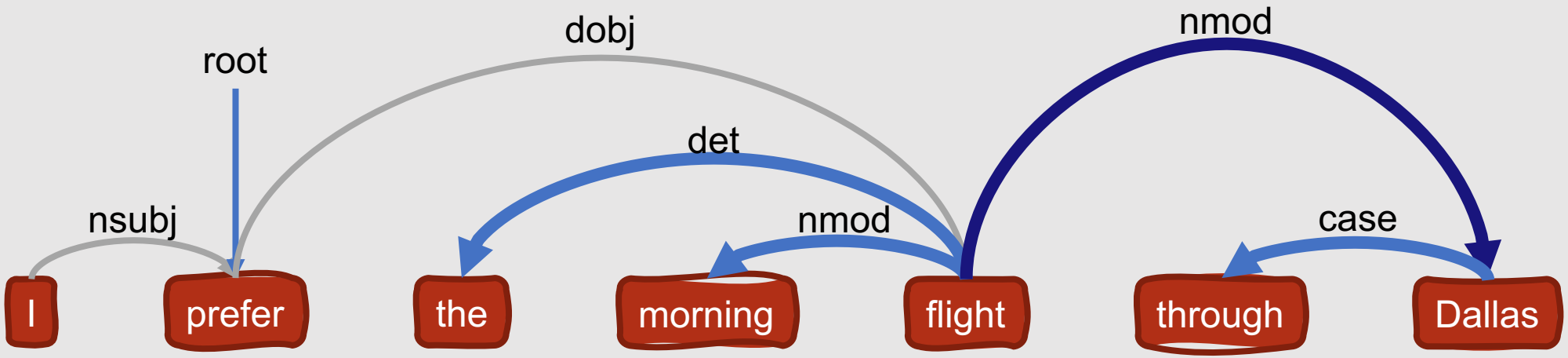
Recently, most researchers have moved toward using universal dependencies.

- Universal dependencies can be broken into two sets
  - **Clausal Relations:** Describe syntactic roles with respect to predicates (the part(s) of the sentence that say something about the subject)
  - **Modifier Relations:** Describe the ways that words can modify their heads

Clausal Relations



Modifier Relations



Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj	csubj ccomp xcomp		
Non-Core Dependents of Clausal Predicates	obl vocative expl dislocated	advcl	advmod discourse	aux cop mark
Dependents of Nominals	nmod appos nummod	acl	amod	det case

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj			
Non-Core Dependents of Clausal Predicates	obl vocative expl dislocated			aux cop mark
Dependents of Nominals	nmod appos nummod			det case

Natalie wrote a dissertation.
nsubj(wrote, Natalie)

Natalie wrote a dissertation.
obj(wrote, dissertation)

Natalie wrote him a dissertation.
iobj(wrote, him)

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj			
Non-Core Dependents of Clausal Predicates	obl vocative expl dislocated			aux cop mark
Dependents of Nominals	nmod appos nummod			

Natalie wrote a dissertation for him.
obl(wrote, him)

Natalie, read my dissertation!
vocative(read, Natalie)

It is clear that her dissertation is a masterpiece.
expl(clear, it)

You must not eat it, the dissertation.
dislocated(eat, dissertation)

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj			
Non-Core Dependents of Clausal Predicates	obl vocative expl dislocated			aux cop mark
Dependents of Nominals	nmod appos nummod			det case

The purpose of this dissertation is to determine an optimal strategy for selecting fantasy football teams.
nmod(purpose, dissertation)

My advisor, Natalie, is boring me.
appos(advisor, Natalie)

Yesterday she wrote five dissertations.
nummod(dissertations, five)

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj	csubj ccomp xcomp		
Non-Core Dependents of Clausal Predicates	obl vocative expl dislocated	advcl		aux
Dependents of Nominals	nmod appos nummod	acl		

What she said about only choosing fantasy football players from northern teams makes sense.
csubj(makes, said)

She said to read her second dissertation for more tips.
ccomp(said, read)

I consider her a genius.
xcomp(consider, genius)

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj	csubj ccomp xcomp		
Non-Core Dependents of Clausal Predicates	obl vocative expl dislocated	advcl	advmod discourse	aux cop mark
Dependents of Nominals	nmod appos nummod	acl	amod	det case

He was upset when she read her dissertation to him.
advcl(upset, read)

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj	csubj ccomp xcomp		
Non-Core Dependents of Clausal Predicates	obl vocative expl dislocated	advcl		mark
Dependents of Nominals	nmod appos nummod	acl	amod	det case

She added an appendix discussing fantasy bachelor.
acl(appendix, discussing)

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	He was deeply offended by the generalization from football to reality show. **advmod(offended, deeply)**			
Non-Core Dependents of Clausal Predicates	obl vocative expl dislocated	advcl	advmod discourse	aux cop mark
Dependents of Nominals	She said, “Well, I have other dissertations that explain this further.” **discourse(explain, well)**			det case

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj	csubj ccomp xcomp		
Non-Core Dependents of Clausal Predicates			advmod discourse	aux cop mark
Dependents of Nominals	nmod appos nummod	acl	amod	det case

He sat with rapt attention as she began reading an excerpt from her third dissertation.
amod(attention, rapt)

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates				
Non-Core Dependents of Clausal Predicates				aux cop mark
Dependents of Nominals				det case

She had included case studies.
aux(included, had)

The studies demonstrated that both fantasy games were essentially the same.
cop(same, were)

He knew that this would change life as he knew it.
mark(change, that)

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

Universal Dependencies

Structural categories of dependent

Functional categories w.r.t. head

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj	csubj		
Non-Core Dependents of Clausal Predicates	obl vocative	advcl	advmod discourse	aux cop mark
Dependents of Nominals	nmod appos nummod	acl	amod	det case

She told him that was the point.
det(point, the)

Then she waved goodbye and went to her office.
case(office, to)

Other miscellaneous dependency relations (see https://universaldependencies.org/u/dep/index.html for details):
conj, cc, fixed, flat, compound, list, parataxis, orphan, goeswith, reparandum, punct, root, dep

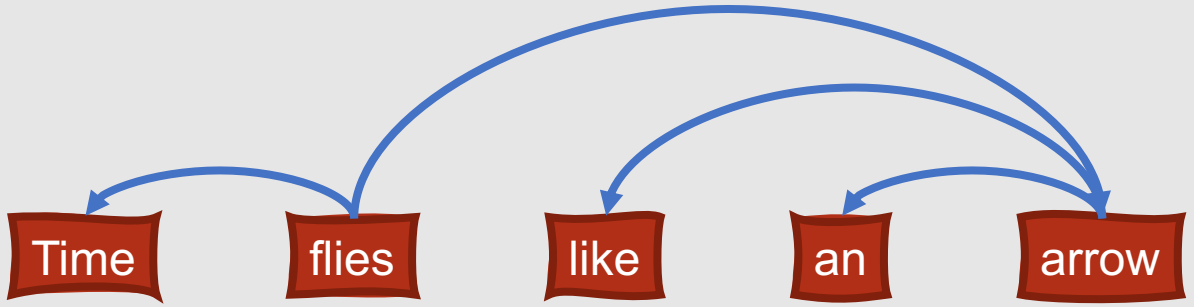
In-Class Exercise

- Assign universal dependency relations to the following sentences:
  - **Time flies like an arrow.**
  - **Fruit flies like a banana.**

https://www.google.com/search?q=timer

	Nominals	Clauses	Modifier Words	Function Words
Core Arguments of Clausal Predicates	nsubj obj iobj	csubj ccomp xcomp		
Non-Core Dependents of Clausal Predicates	obl vocative expl dislocated	advcl	advmod discourse	aux cop mark
Dependents of Nominals	nmod appos nummod	acl	amod	det case

In-Class Exercise



nsubj nmod case det compound obj

Dependency Formalisms

- Dependency structures are directed graphs
  - $G = (V, A)$
    - V is a set of vertices
    - A is a set of ordered pairs of vertices, or arcs
  - V corresponds to the words in a sentence
    - May also include punctuation
    - In morphologically complex languages, may include stems and affixes
  - Arcs capture the grammatical relationships between those words
- According to most grammatical theories (including those followed in this course), dependency structures:
  - Must be connected
  - Must have a designated root node
  - Must be acyclic

Dependency Trees

- Directed graphs (such as those we've seen already) that satisfy the following constraints:
  - Single designated root node
    - No incoming arcs to the root!
  - All vertices *except the root node* have exactly one incoming arc
  - There is a unique path from the root node to each vertex

How to translate from constituent to dependency structures?

Two steps:

- Identify all head-dependent relations in the constituent tree
- Identify the correct dependency relations for those relations

One algorithm for doing this:

- Mark the **head child** of each node in a phrase structure, based on a set of predetermined rules
- In the dependency structure, make the head of each **non-head child** depend on the head of the **head child**

However, doing this can produce results that are far from perfect!

- Most noun phrases do not have much (or any) internal structure
- Morphological information has little to no presence in phrase structure trees
- For **low resource languages** in particular, most dependency treebanks are developed manually by human annotators rather than attempting to automatically translate from constituent to dependency parse

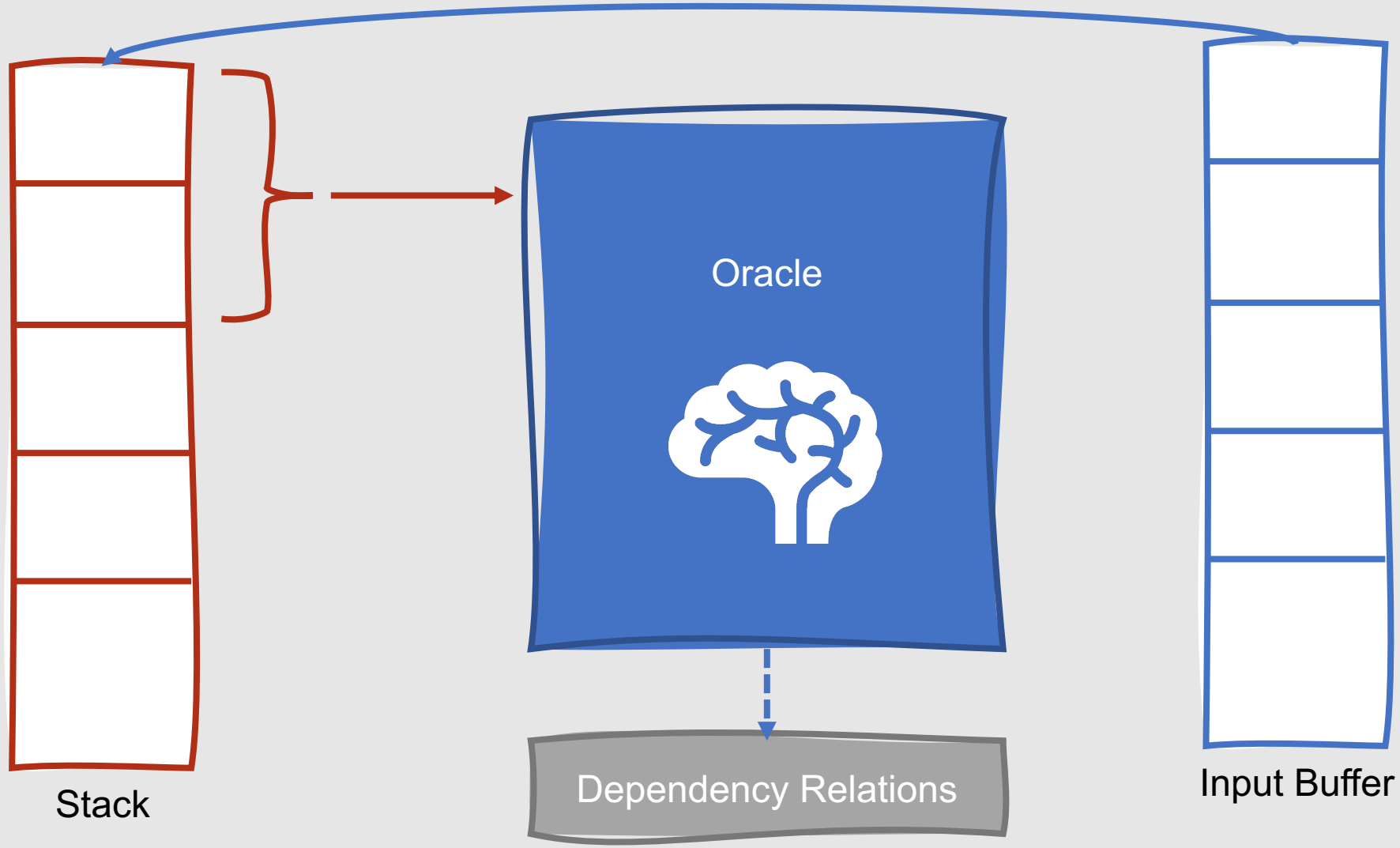
Types of Dependency Parsers

Transition	Transition-based • Build a single tree in a left-to-right (assuming a left-to-right language) sweep over the input sentence
Graph	Graph-based • Search through the space of possible trees for a given sentence, and try to find the tree that maximizes some score

Transition-based Dependency Parsing

- Earliest transition-based approach: **shift-reduce parsing**
  - Input tokens are successively shifted onto a stack
  - The two top elements of the stack are matched against a set of possible relations provided by some knowledge source
  - When a match is found, a head-dependent relation between the matched elements is asserted
- Goal is to find a final parse that accounts for all words

Shift-Reduce Parsing



Transition-based Parsing

- We can build upon shift-reduce parsing by defining a set of **transition operators** to guide the parser's decisions
- Transition operators work by producing new **configurations**:
  - Stack
  - Input buffer of words
  - Set of relations representing a dependency tree

Transition-based Parsing

Initial configuration:

- Stack contains the ROOT node
- Word list is initialized with all words in the sentence, in order
- Empty set of relations represents the parse

Final configuration:

- Stack should be empty
- Word list should be empty
- Set of relations represents the parse

Operators

- The operators used in transition-based parsing then perform the following tasks:
  - Assign the current word as the head of some other word that has already been seen
  - Assign some other word that has already been seen as the head of the current word
  - Do nothing with the current word

Operators

- More formally, these operators are defined as:
  - **LeftArc:** Asserts a head-dependent relation between the word at the top of the stack and the word directly beneath it (the second word), and removes the second word from the stack
    - Cannot be applied when ROOT is the second element in the stack
    - Requires two elements on the stack
  - **RightArc:** Asserts a head-dependent relation between the second word and the word at the top of the stack, and removes the word at the top of the stack
    - Requires two elements on the stack
  - **Shift:** Removes a word from the front of the input buffer and pushes it onto the stack
- These operators implement the **arc standard approach** to transition-based parsing

Arc Standard Approach to Transition-based Parsing

Notable characteristics:

- Transition operators only assert relations between elements at the top of the stack
- Once an element has been assigned its head, it is removed from the stack
  - Not available for further processing!

Benefits:

- Reasonably effective
- Simple to implement

Formal Algorithm: Arc Standard Approach

```
state ← {[root], [words], []}
while state not final:
      # Choose which transition operator to apply
     transition ← oracle(state)

      # Apply the operator and create a new state
     state ← apply(transition, state)
```

When does the process end?

- When all words in the sentence have been consumed
- When the ROOT node is the only element remaining on the stack

Is this another example of dynamic programming?

- No! 😮
- The arc standard approach is a **greedy algorithm**
  - Oracle provides a single choice at each step
  - Parser proceeds with that choice
    - No other options explored
    - No backtracking
  - Single parse returned at the end

Arc Standard: Example

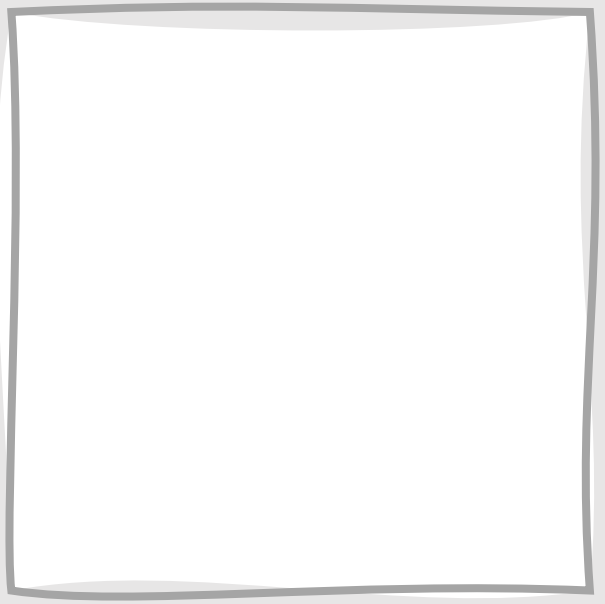
Input Buffer

book	me	the	morning	flight

Stack

root					

Relations



Arc Standard: Example

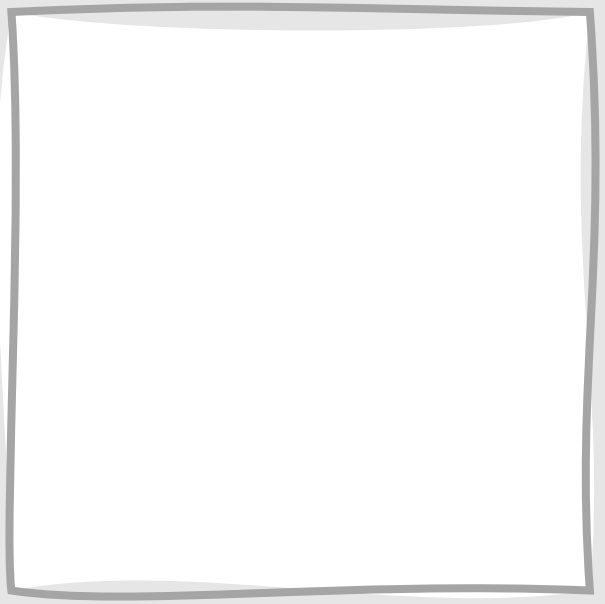
Input Buffer

	me	the	morning	flight

Stack

book	root				

Relations



Only one item in the stack!

Shift **book** from the input buffer to the stack

Arc Standard: Example

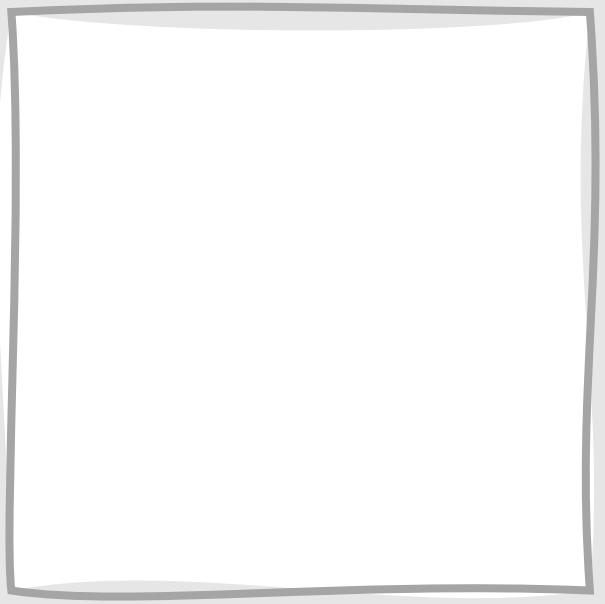
Input Buffer

		the	morning	flight

Stack

me	book	root			

Relations



Valid options: Shift, RightArc

Oracle selects Shift

Shift me from the input buffer to the stack

Arc Standard: Example

Input Buffer

Stack

Relations

(book → me)

Valid options: Shift, RightArc, LeftArc

Oracle selects RightArc

Remove **me** from the stack

Add relation (book → me) to the set of relations

Arc Standard: Example

Input Buffer

			morning	flight

Stack

the	book	root			

Relations

(book → me)

Valid options: Shift, RightArc

Oracle selects Shift

Shift **the** from the input buffer to the stack

Arc Standard: Example

Input Buffer

Stack

Relations

(book → me)

Valid options: Shift, RightArc, LeftArc

Oracle selects Shift

Shift **morning** from the input buffer to the stack

Arc Standard: Example

Input Buffer

Stack

Relations

(book → me)

Valid options: Shift, RightArc, LeftArc

Oracle selects Shift

Shift **flight** from the input buffer to the stack

Arc Standard: Example

Input Buffer

Stack

Relations

(book → me)
(flight → morning)

Valid options: RightArc, LeftArc

Oracle selects LeftArc

Remove **morning** from the stack

Add relation (flight → morning) to the set of relations

Arc Standard: Example

Input Buffer

Stack

Relations

(book → me)
(flight → morning)
(flight → the)

Valid options: RightArc, LeftArc

Oracle selects LeftArc

Remove **the** from the stack

Add relation (flight → the) to the set of relations

Arc Standard: Example

Input Buffer

Stack

Relations

(book → me)
(flight → morning)
(flight → the)
(book → flight)

Valid options: RightArc, LeftArc

Oracle selects RightArc

Remove **flight** from the stack

Add relation (book → flight) to the set of relations

Arc Standard: Example

Input Buffer

Stack

Relations

(book → me)
(flight → morning)
(flight → the)
(book → flight)
(root → book)

Valid options: RightArc

Oracle selects RightArc

Remove **book** from the stack

Add relation (root → book) to the set of relations

Arc Standard: Example

Input Buffer

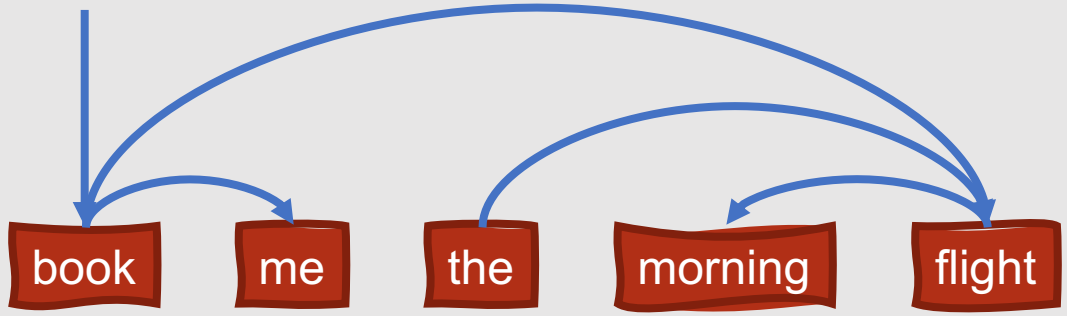
Stack

Valid options: None

State is final

Relations

(book → me)
(flight → morning)
(flight → the)
(book → flight)
(root → book)



A few things worth noting….

- We assumed in the previous example that our oracle was always correct …this is not necessarily (or perhaps not even likely) the case!
  - Incorrect choices lead to incorrect parses since the algorithm cannot perform any backtracking
- Alternate sequences may also lead to equally valid parses

How do we get actual dependency labels?

- Parameterize **LeftArc** and **RightArc**
  - LeftArc(nsubj), RightArc(obj), etc.
- Of course, this makes the oracle's job more difficult (much larger set of operators from which to choose!)

iobj(book → me)
compound(flight → morning)
det(flight → the)
obj(book → flight)
root(root → book)

How does the oracle know what to choose?

- State of the art systems use **supervised machine learning** for this task
- This requires a training set of configurations labeled with correct transition operators
- The person designing the system needs to decide what types of features should be extracted from these configurations to best train the oracle (a machine learning model)
- The oracle will then learn which transitions to predict for previously-unseen configurations based on the extracted features and associated labels for configurations in the training set

What types of machine learning models are used as oracles?

Commonly:

- Logistic regression
- Support vector machines

Recently:

- Neural networks

Graph-based Dependency Parsing

- Search through the space of possible trees for a given sentence, attempting to maximize some score
- This score is generally a function of the scores of individual subtrees within the overall tree
- **Edge-factored approaches** determine scores based on the scores of the edges that comprise the tree
  - overall_score(t) = $\sum_{e \in t} score(e)$
  - Letting *t* be a tree for a given sentence, and *e* be its edges

Why use graph-based methods for dependency parsing?

- Transition-based methods tend to have high accuracy on shorter dependency relations, but that accuracy declines as the distance between the two words increases
- This is largely due to the fact that transition-based methods are greedy …they can be fooled by seemingly-optimal local solutions
- Graph-based methods score entire trees, thereby avoiding that issue

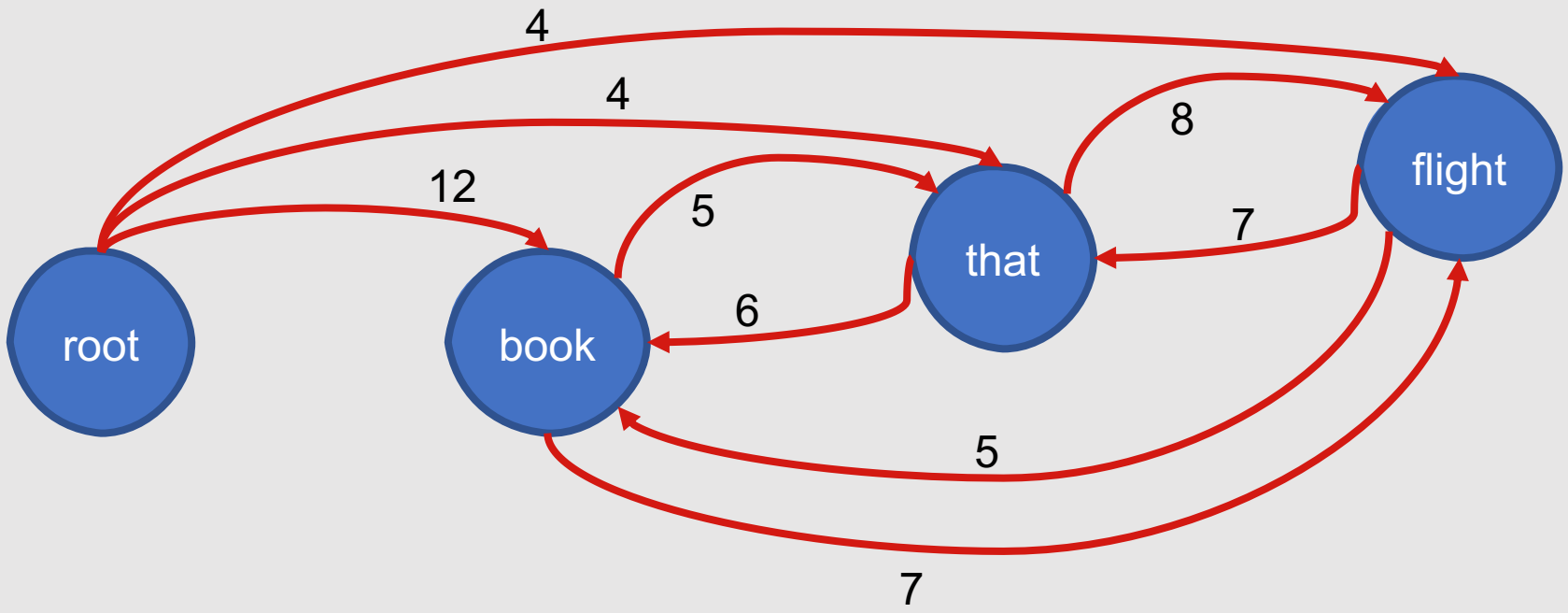
Maximum Spanning Tree

Given an input sentence, construct a fully-connected, weighted, directed graph

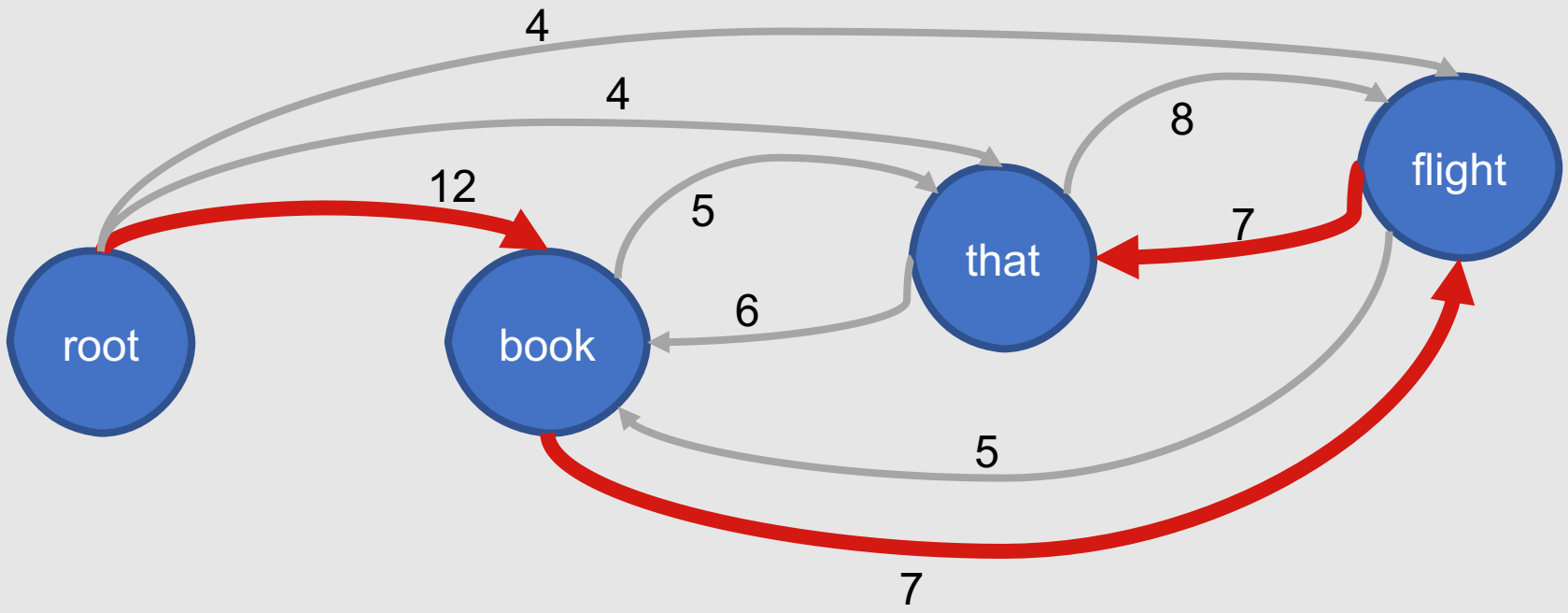
- Vertices are input words
- Directed edges represent all possible head-dependent assignments
- Weights reflect the scores for each possible head-dependent assignment, predicted by a supervised machine learning model

A maximum spanning tree represents the preferred dependency parse for the sentence, as determined by the weights

Maximum Spanning Tree: Example



Maximum Spanning Tree: Example



Two intuitions to keep in mind….

- Every vertex in a spanning tree has exactly one incoming edge
- Absolute values of the edge scores are not critical
  - Relative weights of the edges entering a vertex are what matter!

How do we know that we have a spanning tree?

- Given a fully-connected graph G = (V, E), a subgraph T = (V, F) is a spanning tree if:
  - It has no cycles
  - Each vertex (except the root) has exactly one edge entering it

How do we know that we have a maximum spanning tree?

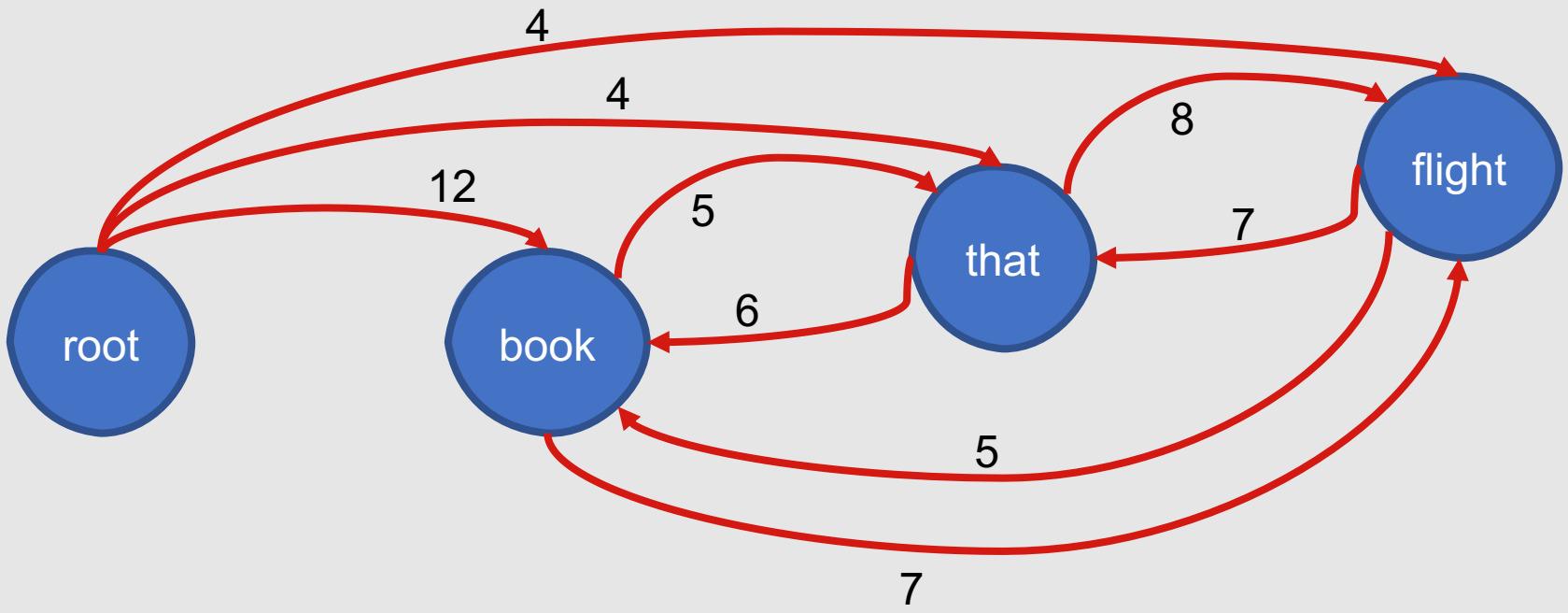
- **If the greedy selection process produces a spanning tree, then that tree is the maximum spanning tree**
- However, the greedy selection process may select edges that result in cycles
- If this happens, an alternate graph can be created that collapses cycles into new nodes, with edges that previously entered or exited the cycle now entering or exiting the new node
- The greedy selection process is then recursively applied to the new graph until a (maximum) spanning tree is found

Formal Algorithm

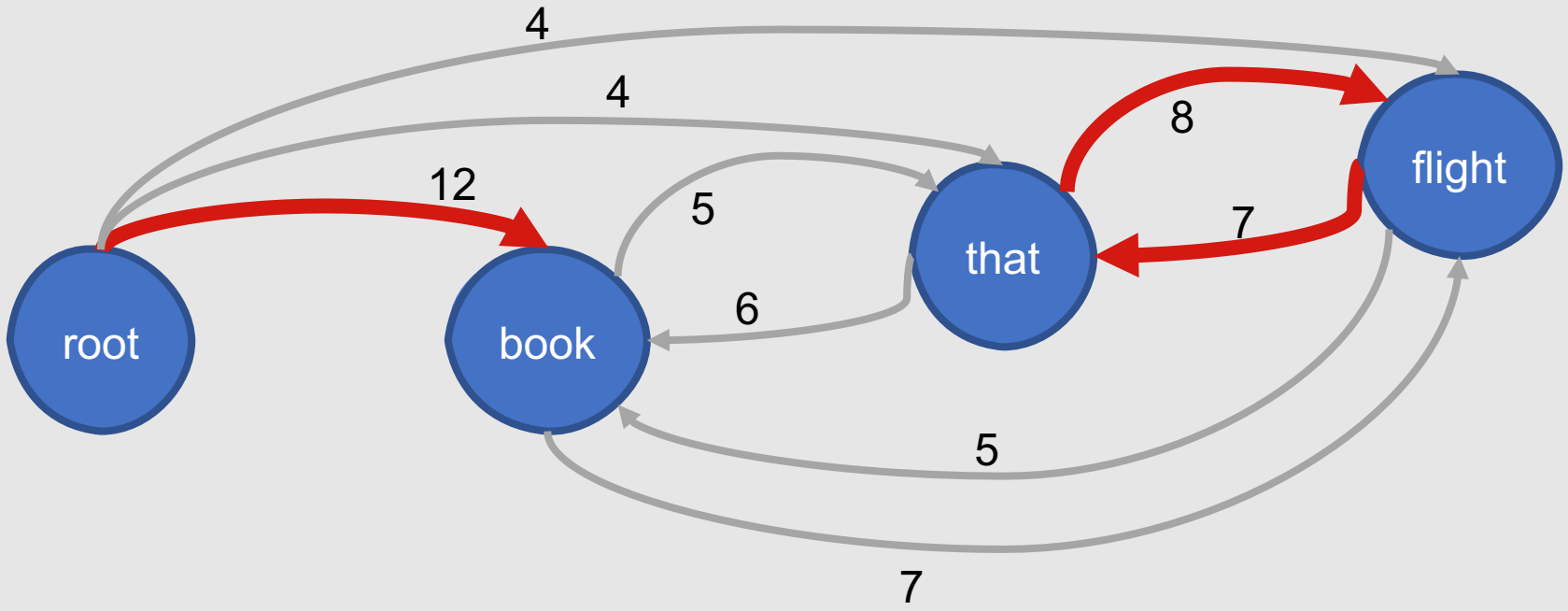
```
F ← []
T ← []
score' ← []
for each v in V do:
        bestInEdge ← argmax_{e=(u,v)∈E} score[e]
        F ← F U bestInEdge
        for each e = (u,v) ∈ E do:
                score'[e] ← score[e] - score[bestInEdge]

        if T=(V,F) is a spanning tree:
                return T
        else:
                C ← a cycle in F
                G' ← collapse(G, C)
                T' ← maxspanningtree(G', root, score') # Recursively call the current function
                T ← expand(T', C)
                return T
```

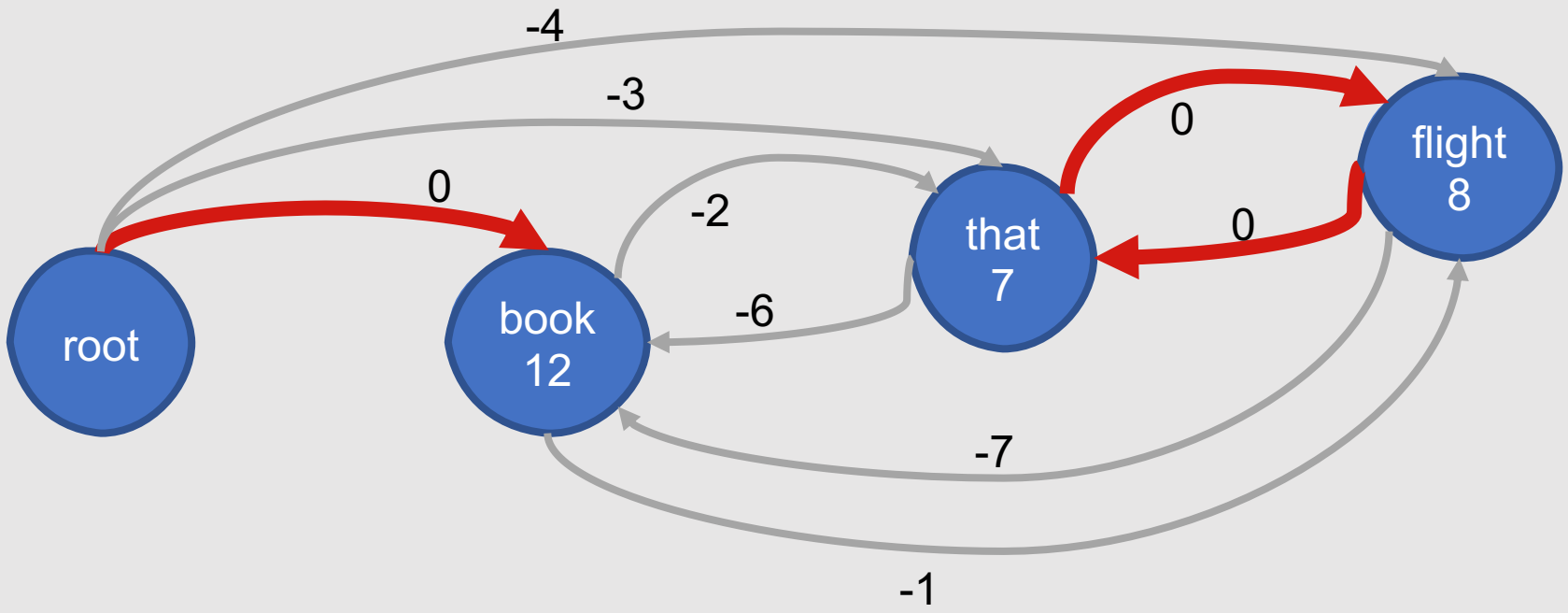
Maximum Spanning Tree: Updated Example



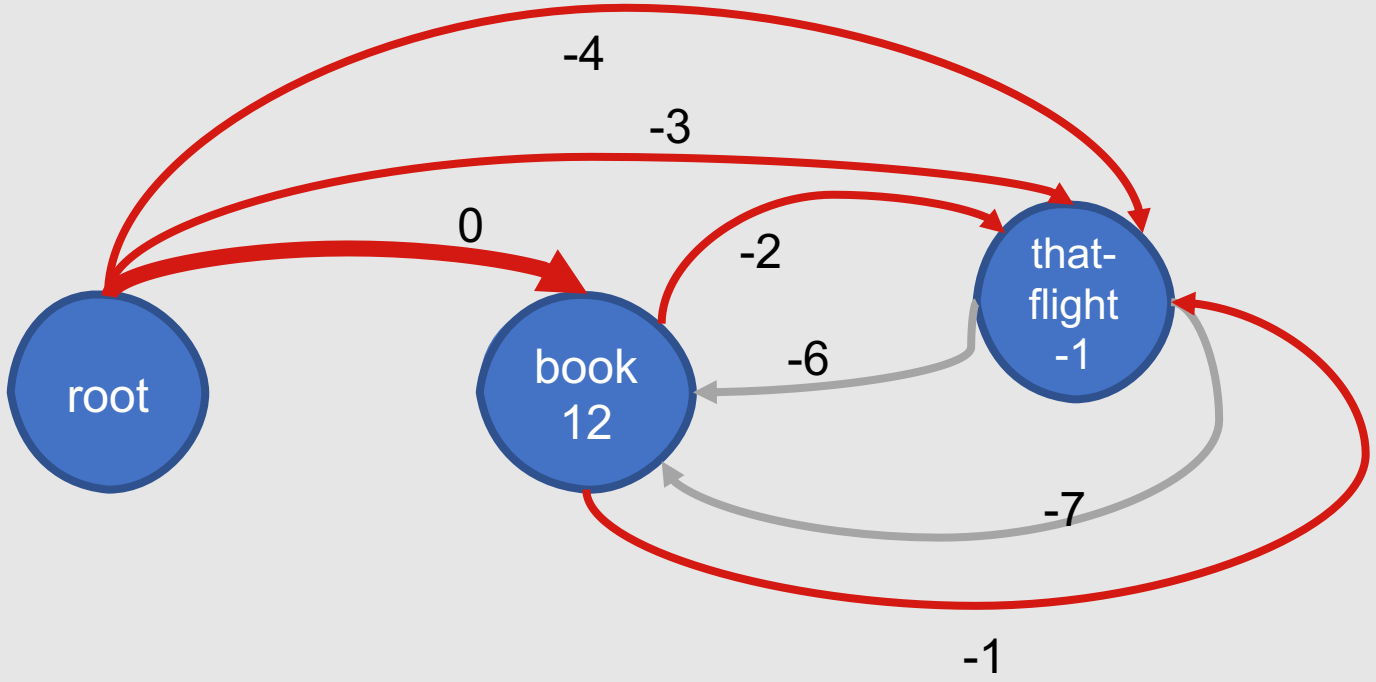
Maximum Spanning Tree: Updated Example



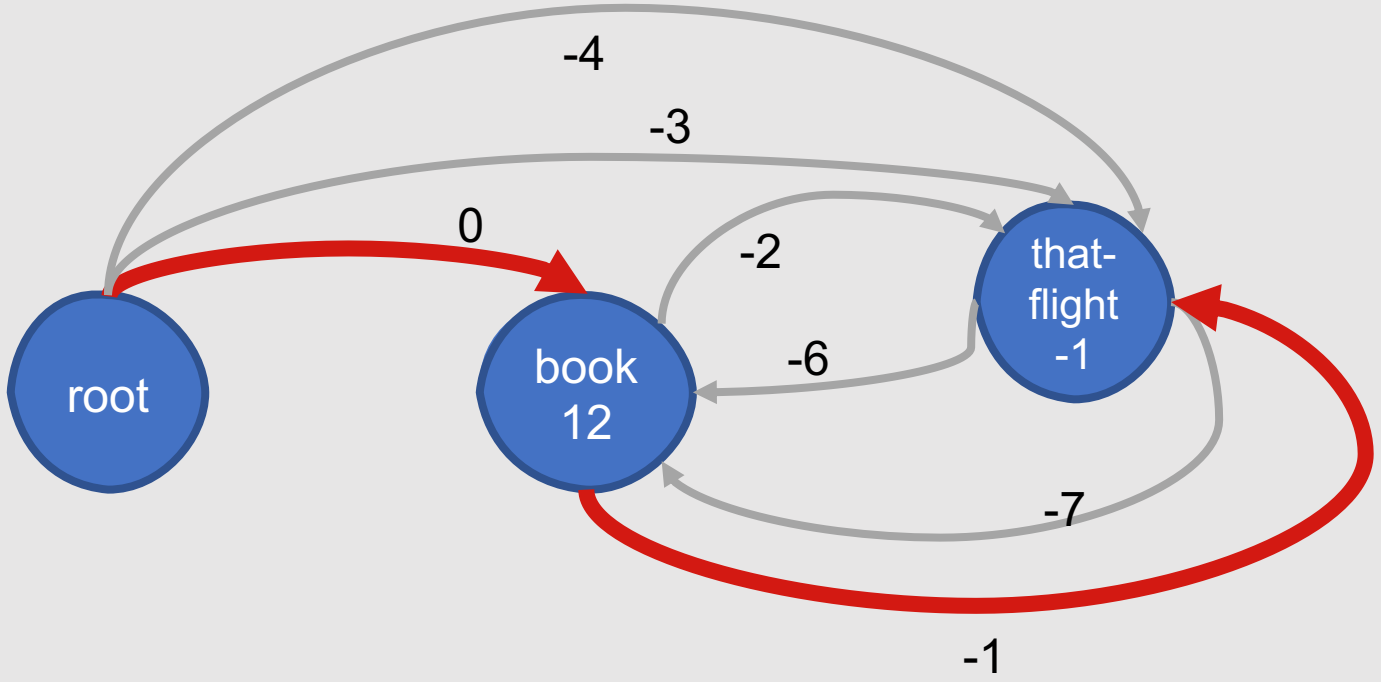
Maximum Spanning Tree: Updated Example



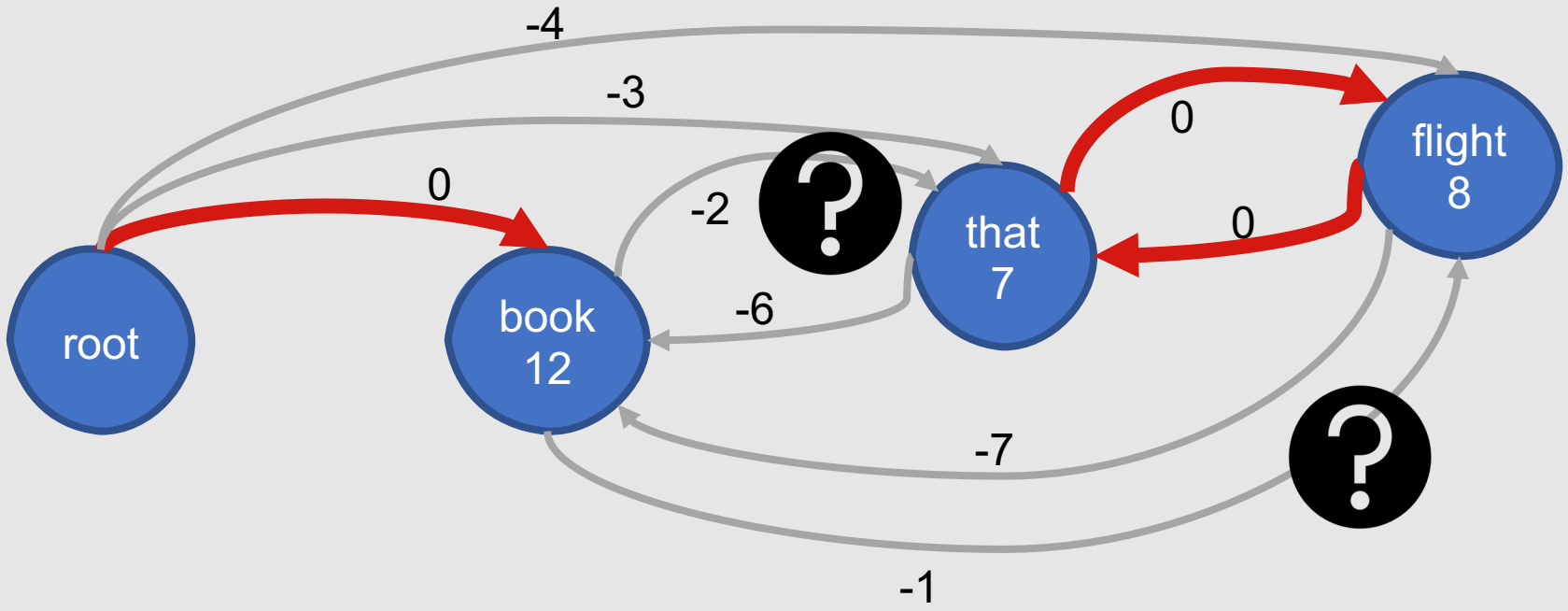
Maximum Spanning Tree: Updated Example



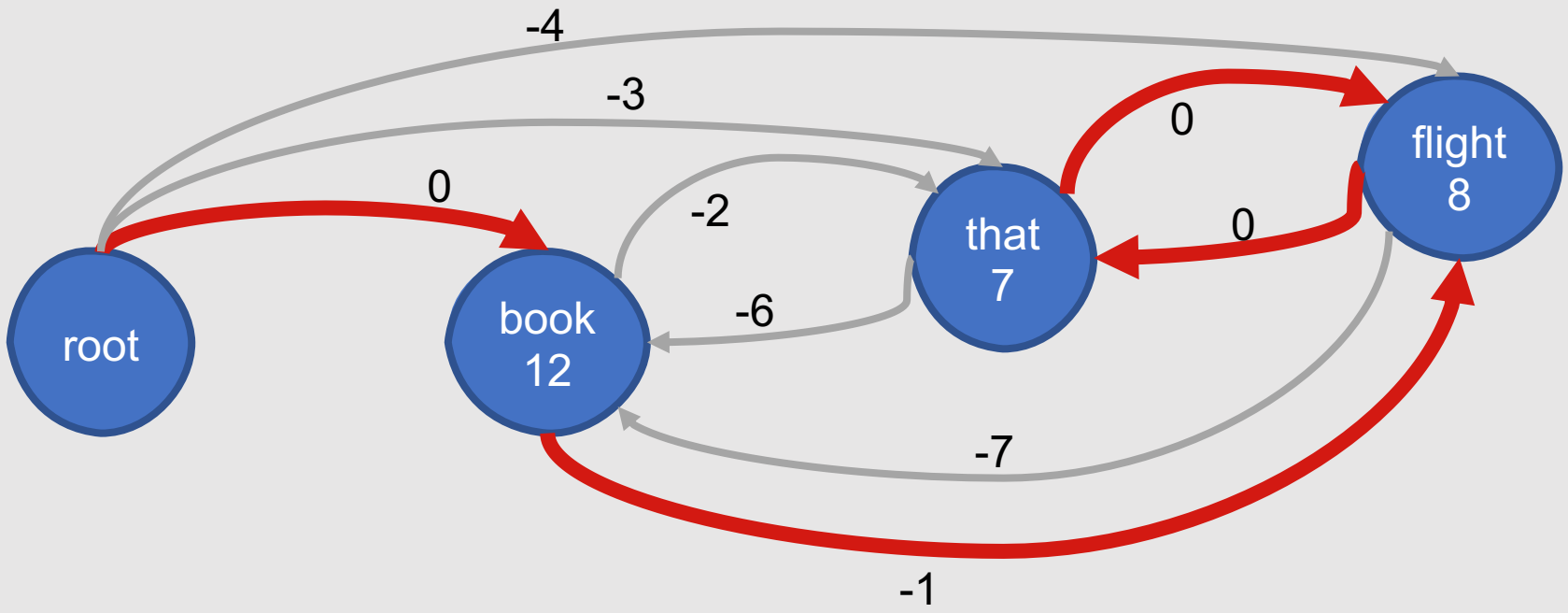
Maximum Spanning Tree: Updated Example



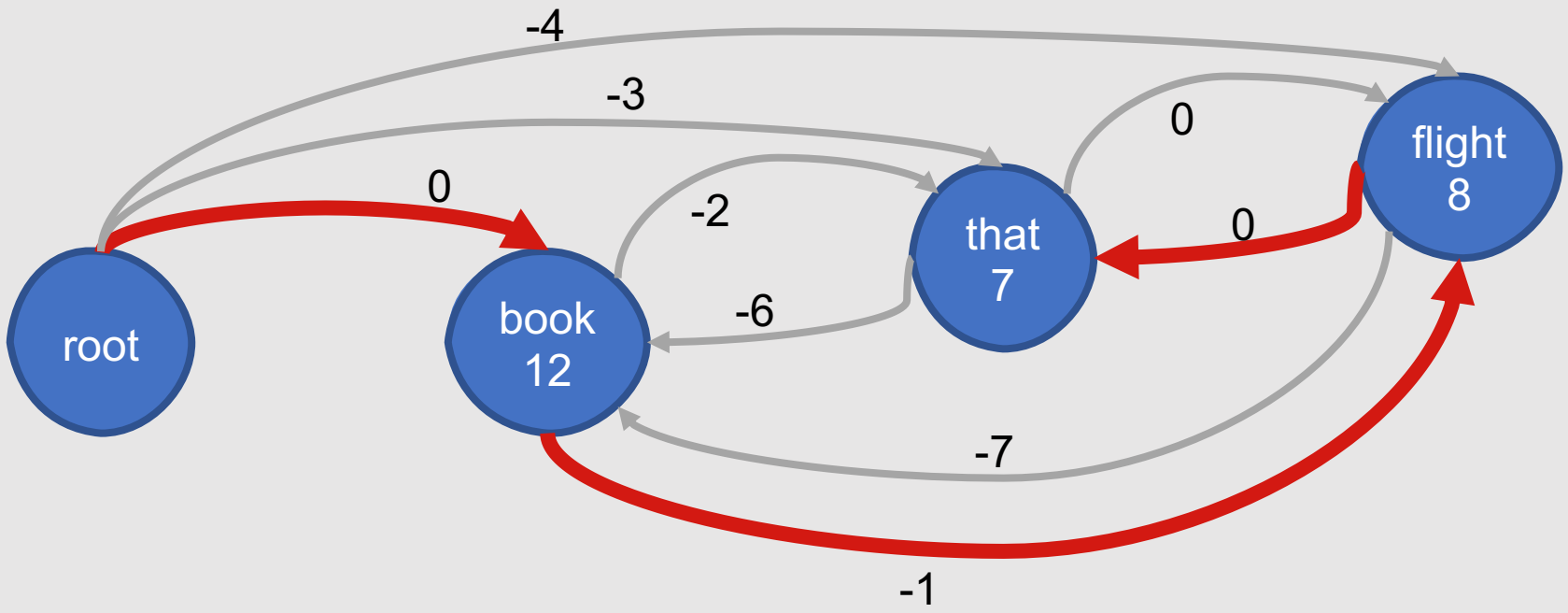
Maximum Spanning Tree: Updated Example



Maximum Spanning Tree: Updated Example



Maximum Spanning Tree: Updated Example



How do we train our model to predict edge weights?

- Similar approach to training the oracle in a transition-based parser
- Common features can include:
  - Words, lemmas, parts of speech
  - Corresponding features from contexts before and after words
  - Word embeddings
  - Dependency relation type
  - Dependency relation direction
  - Distance from head to dependent

Summary: Dependency Parsing

- **Dependency parsing** is the process of automatically determining **directed relationships between words** in a source sentence
- Many dependency taxonomies exist, but the most common taxonomy for English text is the set of **universal dependencies**
- Dependency parsers can be **transition-based** or **graph-based**
- A popular transition-based method is the **arc standard** approach
- A popular graph-based method is the **maximum spanning tree** approach
- Both make use of **supervised machine learning** to aid the decision-making process