CKY Parsing & CNF Conversion

LING 571 — Deep Processing Techniques for NLP
October 7, 2020
Shane Steinert-Threlkeld

Announcements

- **HW #1** due tonight at **11:00pm**.
- If you want to use `python3.6` on Patas:
  - **`/opt/python-3.6/bin/python3`**
  - `nltk` is installed.
- [For personal projects, but not 571 HW, you can use the latest of everything via Anaconda (download with `wget`).]
- When in doubt, use *full paths* for everything (python binary, file names, etc)

Type Hinting in Python

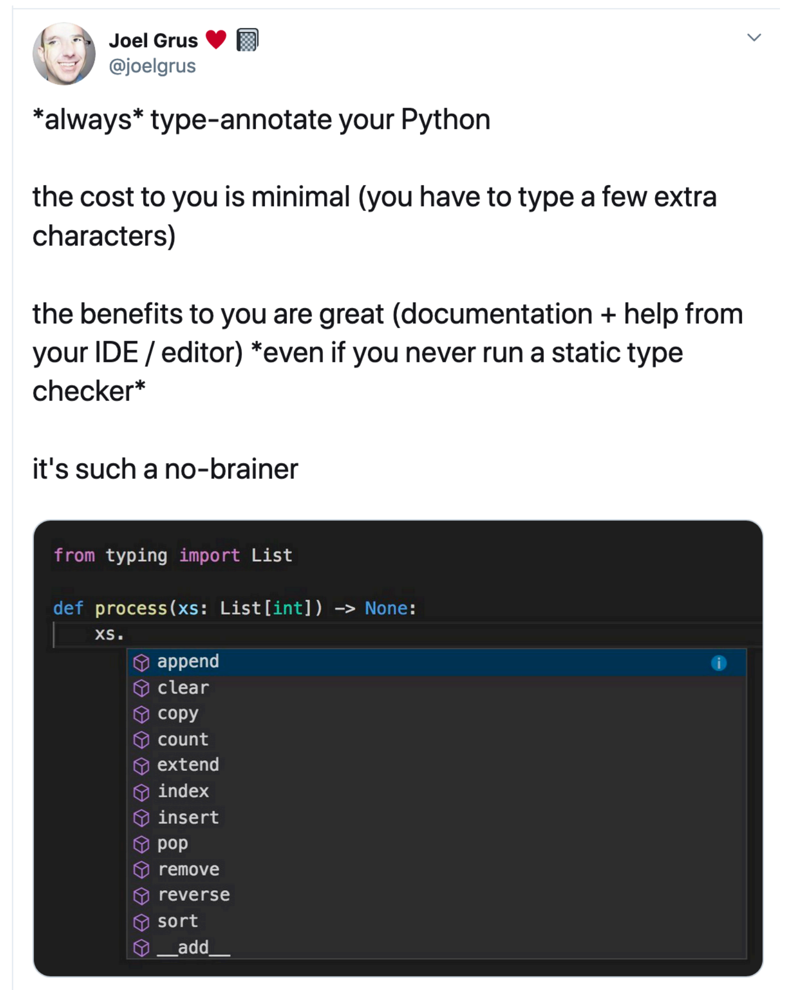
- Supported in ≥3.6 [tutorial]

```
from typing import List
from nltk.grammar import Production

def fix_hybrid_production(hybrid_prod: Production) -> List[Production]:
    …
```

- Also available in PyCharm through docstrings and/or comments:

```
def fix_hybrid_productions(hybrid_prod):
    """
    This function takes a hybrid production and
    returns a list of new CNF productions
    :type hybrid_prod: Production
    :rtype: list[Production]
    """
```



Joel Grus
@joelgrus

always type-annotate your Python

the cost to you is minimal (you have to type a few extra characters)

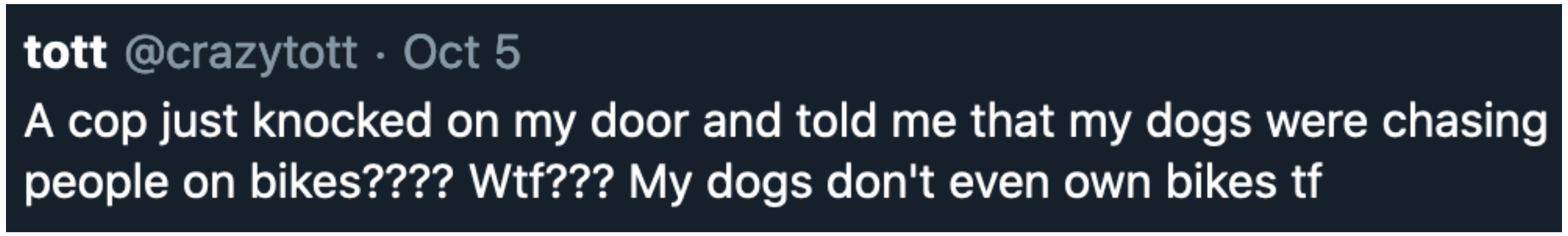
the benefits to you are great (documentation + help from your IDE / editor) *even if you never run a static type checker*

it's such a no-brainer

```
from typing import List

def process(xs: List[int]) -> None:
    xs.
```

Joke of the Week (PP Attachment Ambiguity)


tott @crazytott · Oct 5
A cop just knocked on my door and told me that my dogs were chasing people on bikes???? Wtf??? My dogs don't even own bikes tf

Roadmap

- **Parsing-as-Search**
- Parsing Challenges
- Strategy: Dynamic Programming
- Grammar Equivalence
- CKY parsing algorithm

Computational Parsing

- Given a body of (annotated) text, how can we derive the grammar rules of a language, and employ them in automatic parsing?
  - Treebanks & PCFGs
- Given a grammar, how can we derive the analysis of an input sentence?
  - Parsing as search
  - CKY parsing
    - Conversion to CNF

What is Parsing?

- CFG parsing is the task of assigning trees to input strings
  - For any input ***A*** and grammar ***G***
    - …assign ≥0 parse trees ***T*** that represent its syntactic structure, and…
    - Cover all and only the elements of ***A***
    - Have, as root, the start symbol ***S*** of ***G***
    - …do not necessarily pick one single (or correct) analysis
- Subtask: Recognition
  - Given input ***A***, ***G*** – is ***A*** in language defined by ***G*** or not?

Motivation

- Is this sentence in the language — i.e. is it “grammatical?”
  - ** I prefer United has the earliest flight.*
  - FSAs accept regular languages defined by finite-state automata.
  - Our parsers accept languages defined by CFG (equiv. pushdown automata).
- What is the syntactic structure of this sentence?
  - *What airline has the cheapest flight?*
  - *What airport does Southwest fly from near Boston?*
  - Syntactic parse provides framework for semantic analysis
    - What is the subject? Direct object?

Parsing as Search

- Syntactic parsing searches through possible trees to find one or more trees that derive input
- Formally, search problems are defined by:
  - Start state ***S***
  - Goal state ***G*** (with a test)
  - Set of actions that transition from one state to another
    - “Successor function”
  - A path cost function

Parsing as Search: One Model

- Start State ***S***: Start Symbol
- Goal test:
  - Does the parse tree cover all of, and only, the input?
- Successor function:
  - Expand a nonterminal using a production where nonterminal is the LHS of the production
- Path cost:
  - …ignored for now.

Parsing as Search: One Model

- Node:
  - Partial solution to search problem (partial parse)
- Search start node (initial state):
  - Input string
  - Start symbol of CFG
- Goal node:
  - Full parse tree: covering all of, and only the input, rooted at ***S***

Search Algorithms

- Depth First
  - Keep expanding nonterminals until they reach words
  - If no more expansions available, back up
- Breadth First
  - Consider all parses that expand a single nonterminal…
  - …then all with two expanded, etc…
- Other alternatives, if have associated path costs.

Parse Search Strategies

- Two constraints on parsing:
  - Must start with the start symbol
  - Must cover exactly the input string
- Correspond to main parsing search strategies
  - Top-down search (Goal-directed)
  - Bottom-up search (Data-driven search)

A Grammar

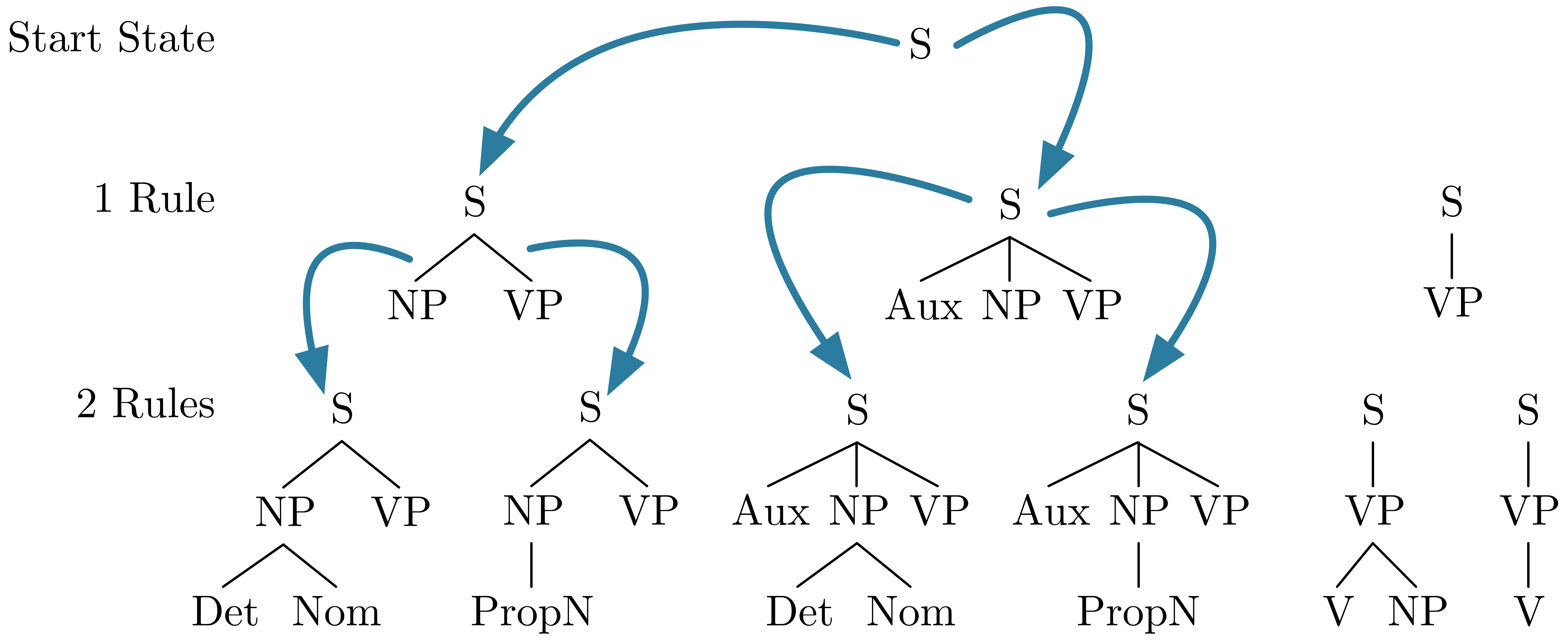
Grammar	Lexicon
S → NP VP	*Det → that | this | a*
S → Aux NP VP	*Noun → book | flight | meal | money*
S → VP	*Verb → book | include | prefer*
NP → Pronoun	*Pronoun → I | she | me*
NP → Proper-Noun	*Proper-Noun → Houston | NWA*
NP → Det Nominal	*Aux → does*
Nominal → Noun	*Preposition → from | to | on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

Jurafsky & Martin, Speech and Language Processing, p.390

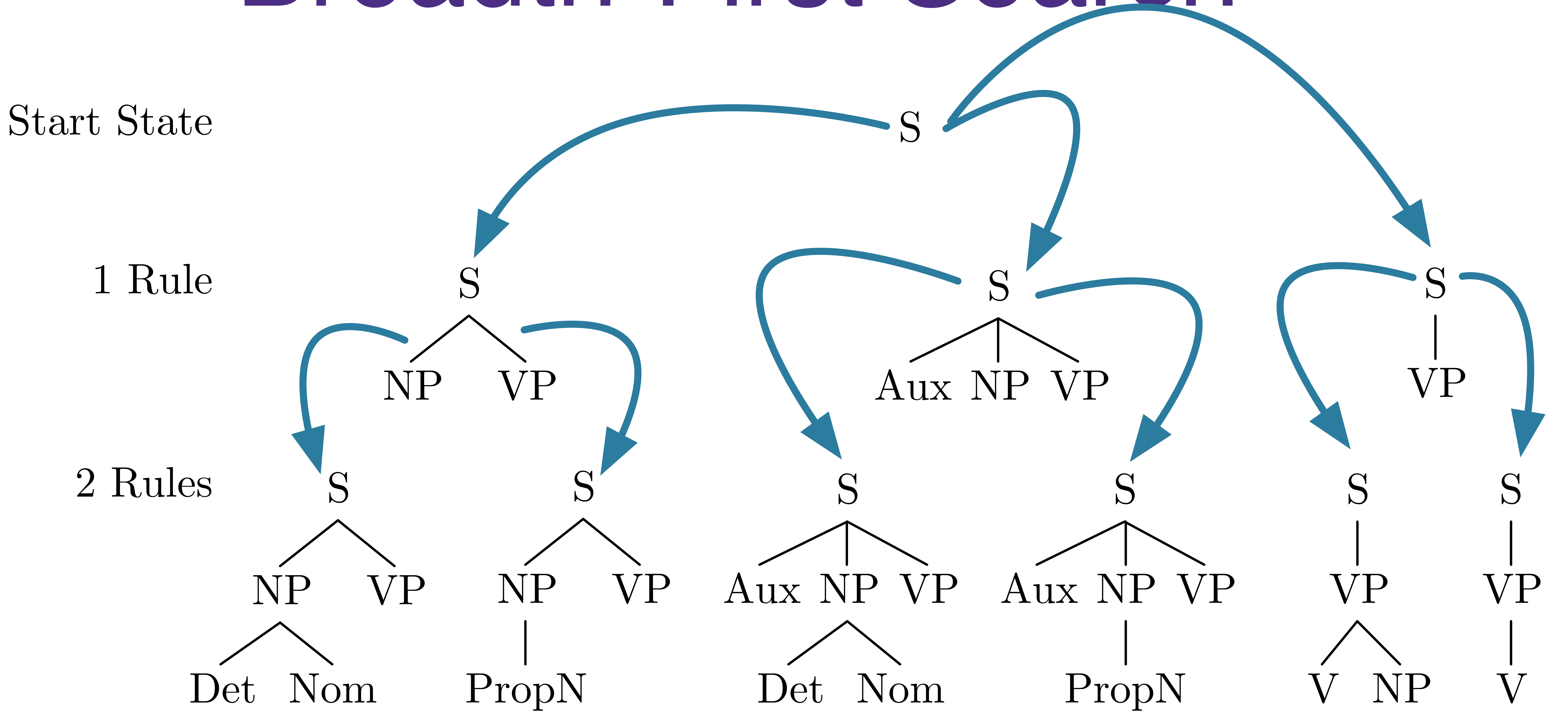
Top-down Search

- All valid parse trees must be rooted with start symbol
- Begin search with productions where S is on LHS
  - e.g. *S* → *NP VP*
- Successively expand nonterminals
  - e.g. *NP* → *Det Nominal*; *VP* → *V NP*
- Terminate when all leaves are terminals

Depth-First Search

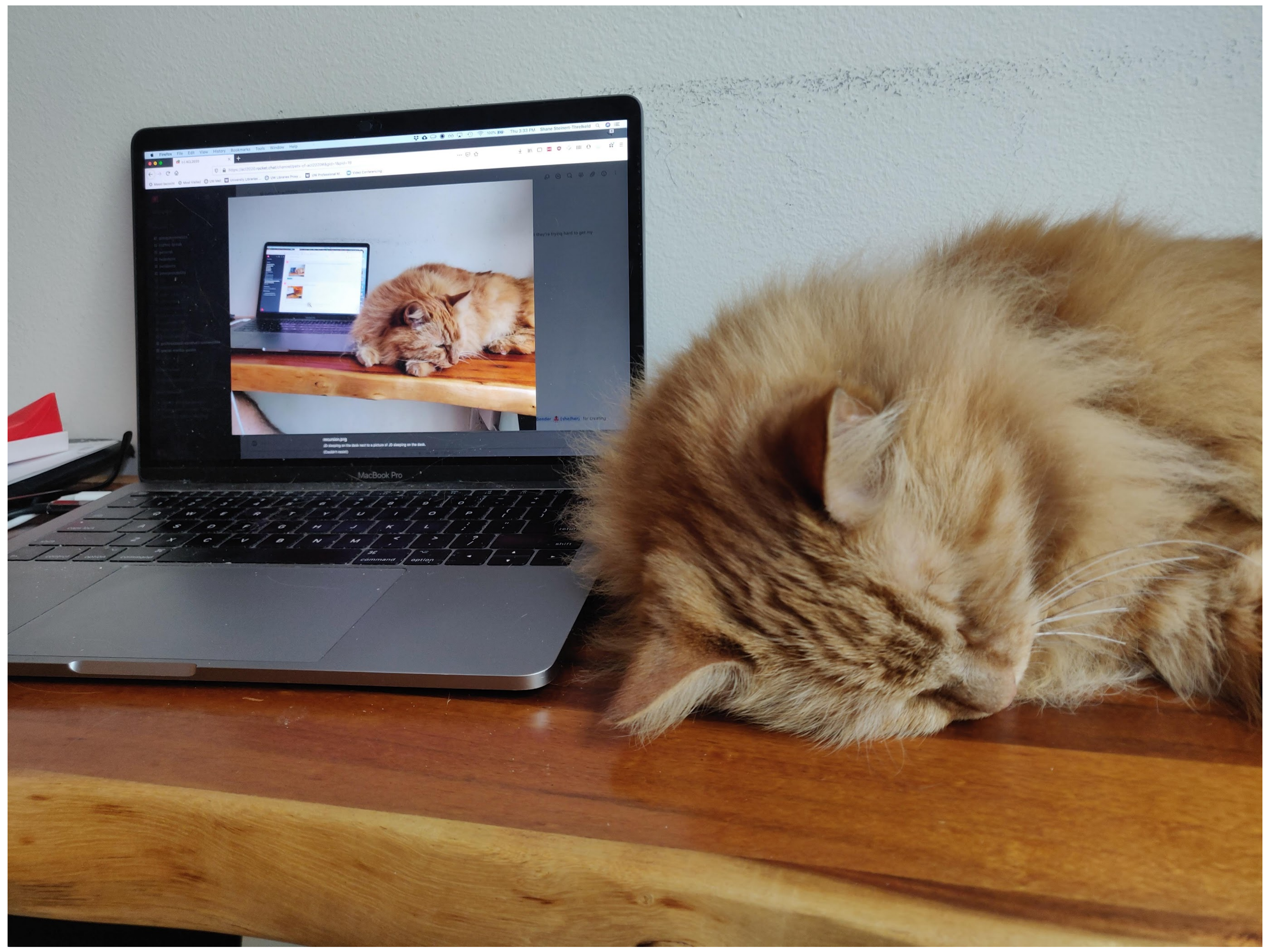


Breadth-First Search



Pros and Cons of Top-down Parsing

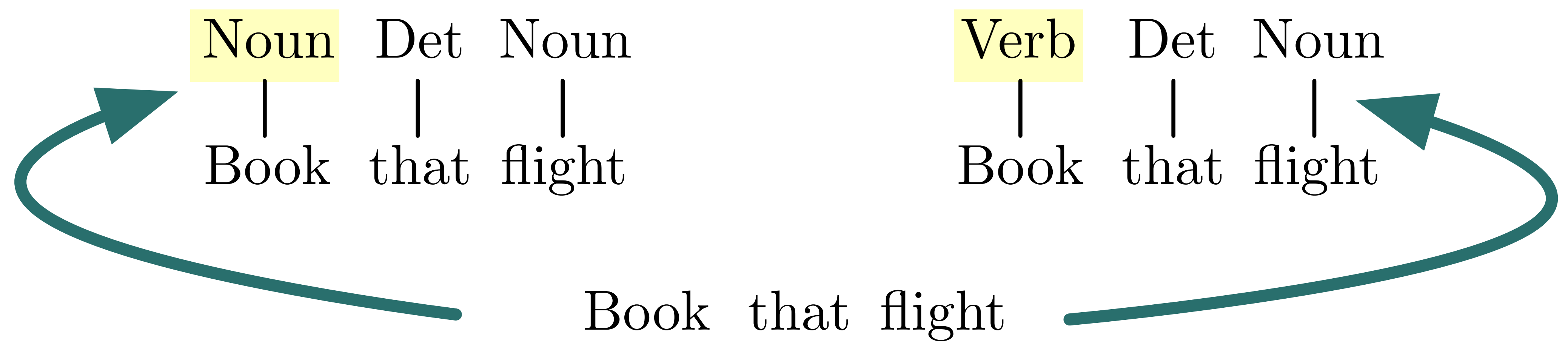
- Pros:
  - Doesn't explore trees not rooted at S
  - Doesn't explore subtrees that don't fit valid trees
- Cons:
  - Produces trees that may not match input
  - May not terminate in presence of recursive rules
  - May re-derive subtrees as part of search

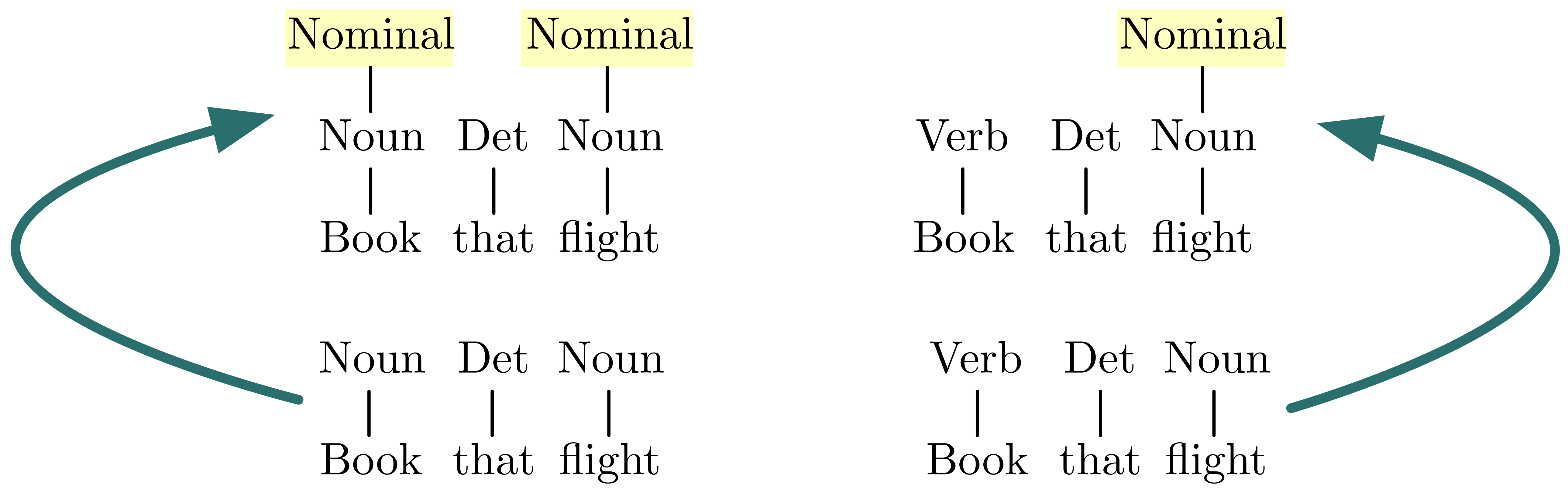


Bottom-Up Parsing

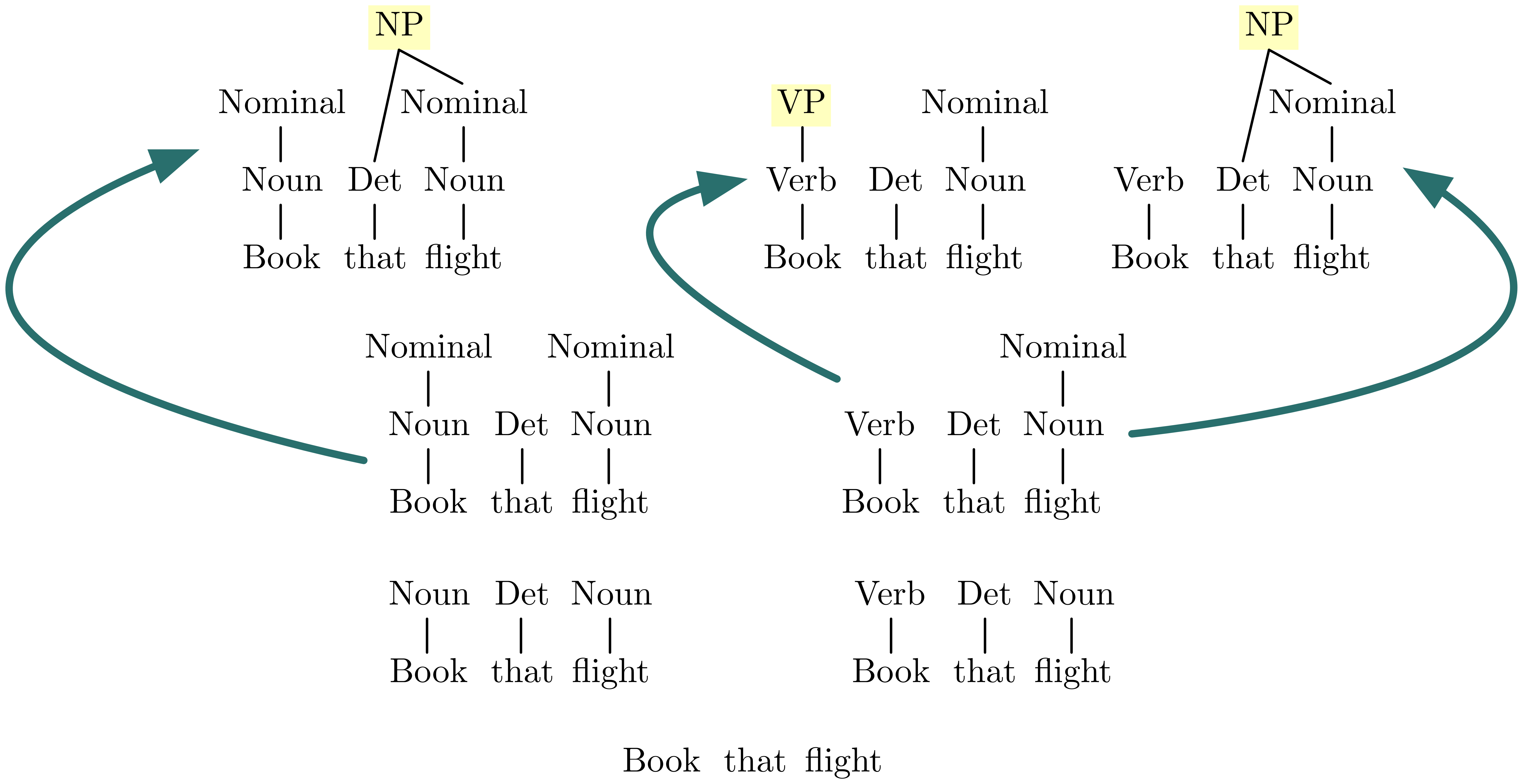
- Try to find all trees that span the input
  - Start with input string
    - Book that flight
- Use all productions with current subtree(s) on RHS
  - e.g. ***N*** → Book; ***V*** → Book
- Stop when spanned by S, or no more rules apply

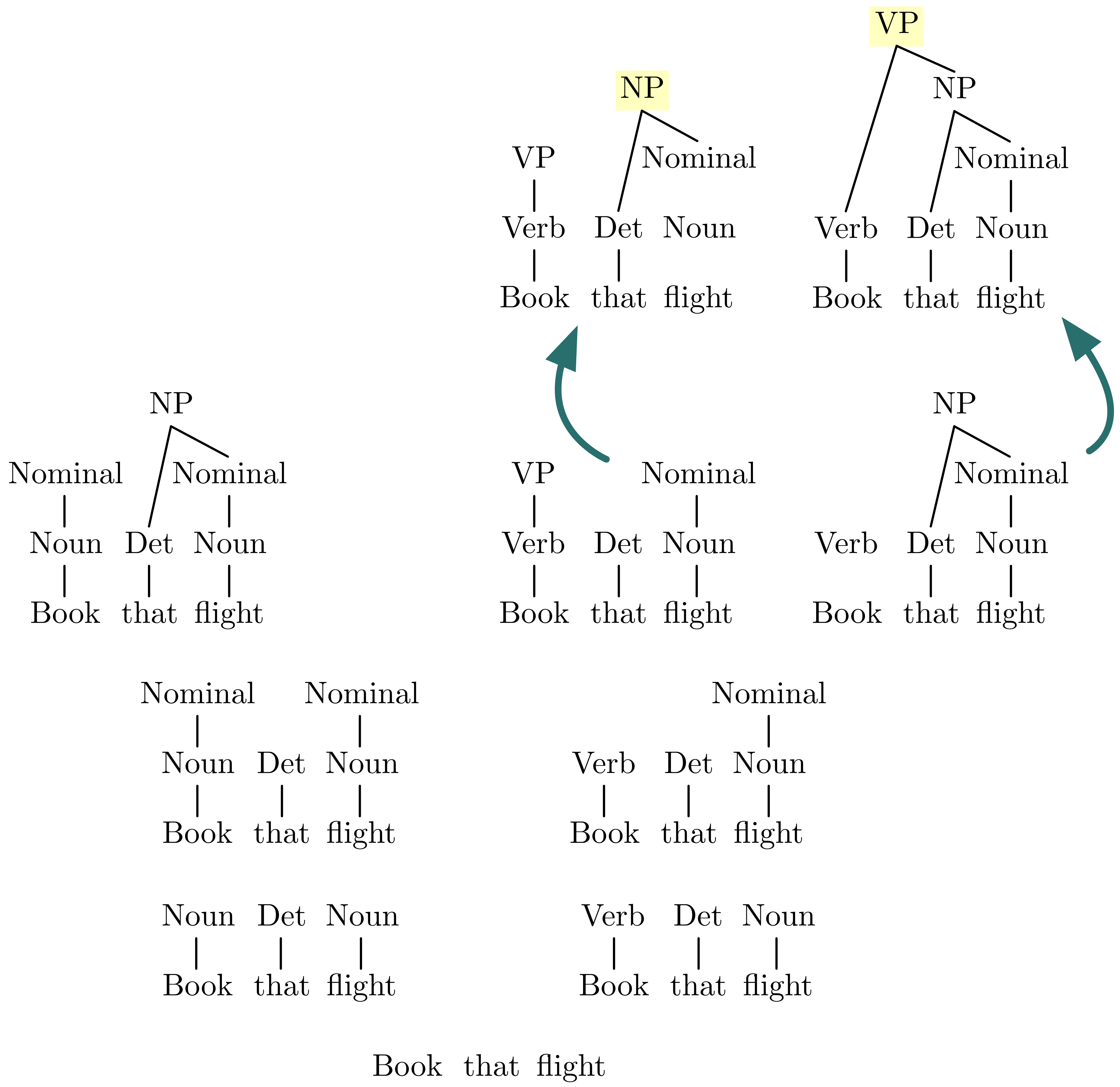
Book that flight


Noun Det Noun
Book that flight
Verb Det Noun
Book that flight
Book that flight



Book that flight

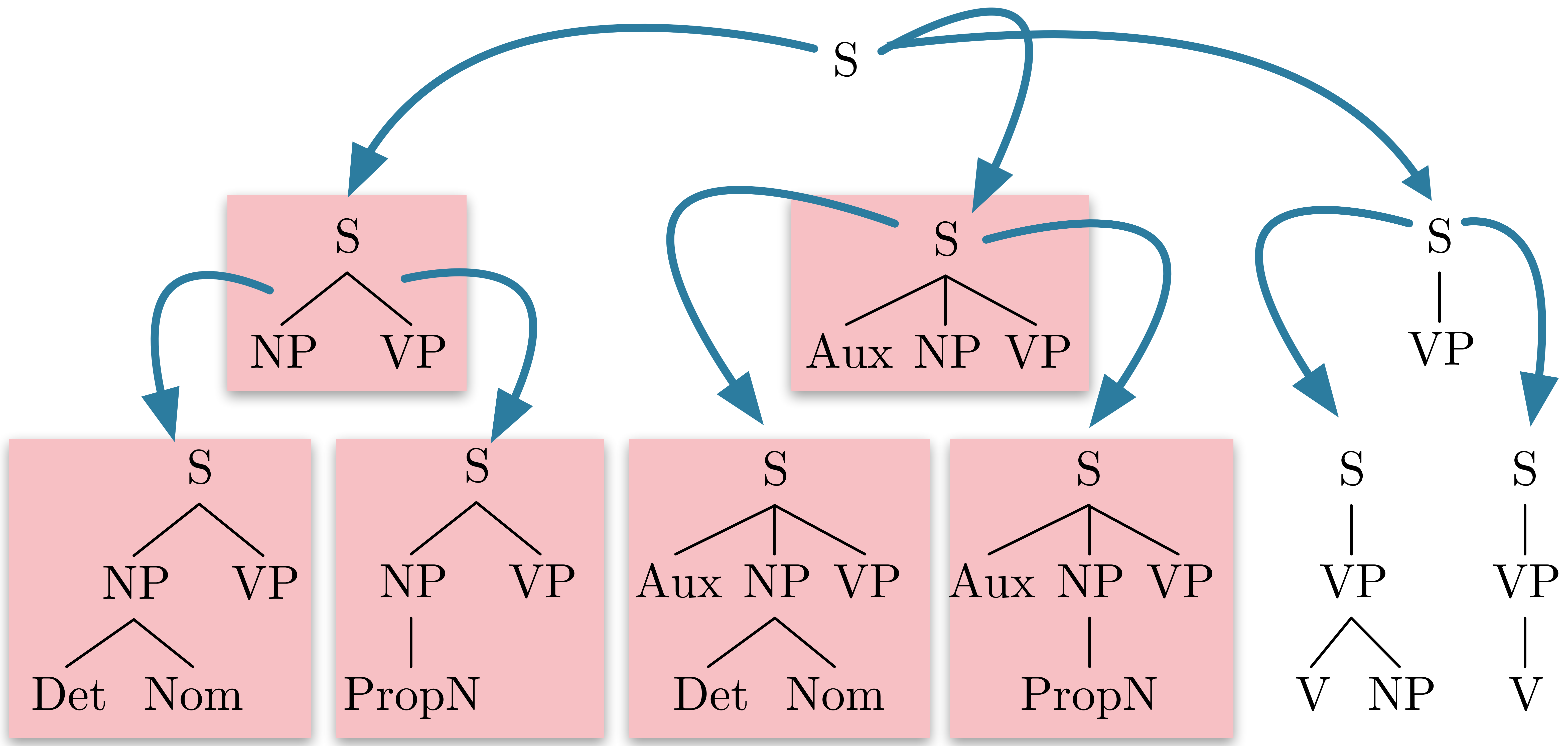

NP
Nominal
Nominal
Noun Det Noun
Book that flight
VP
Nominal
Verb Det Noun
Book that flight
NP
Nominal
Verb Det Noun
Book that flight
Nominal
Nominal
Noun Det Noun
Book that flight
Nominal
Verb Det Noun
Book that flight
Noun Det Noun
Book that flight
Verb Det Noun
Book that flight
Book that flight


VP
NP
NP
VP
Nominal
Nominal
Verb Det Noun
Book that flight
Verb Det Noun
Book that flight
NP
NP
Nominal Nominal
VP Nominal
Nominal
Noun Det Noun
Book that flight
Verb Det Noun
Book that flight
Verb Det Noun
Book that flight
Nominal Nominal
Nominal
Noun Det Noun
Book that flight
Verb Det Noun
Book that flight
Noun Det Noun
Book that flight
Verb Det Noun
Book that flight
Book that flight

Pros and Cons of Bottom-Up Search

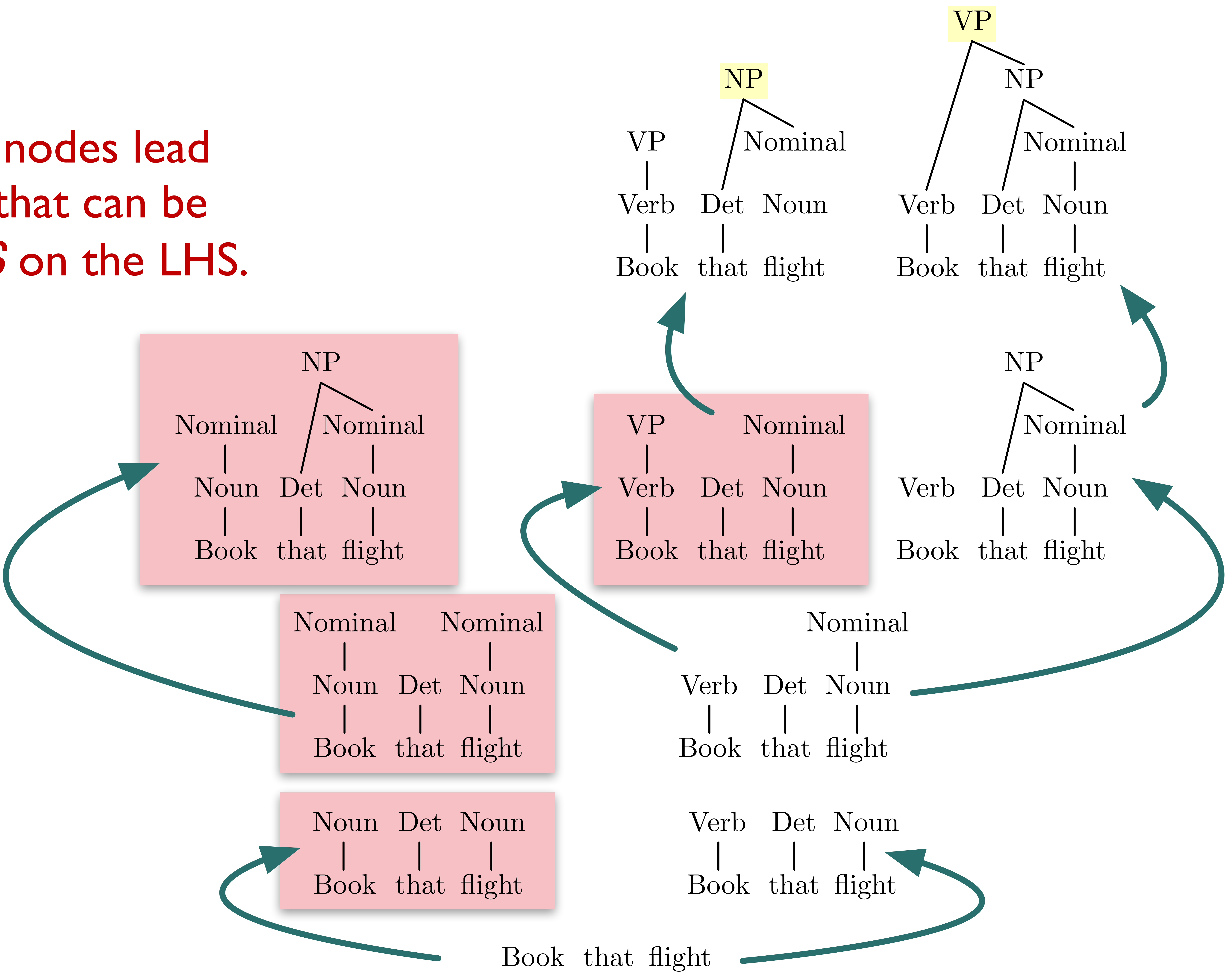
- Pros:
  - Will not explore trees that don't match input
  - Recursive rules less problematic
  - Useful for incremental/fragment parsing
- Cons:
  - Explore subtrees that will not fit full input

Recap: Parsing as Search



None of these nodes can produce *book* as first terminal

None of these nodes lead lead to a RHS that can be combined with *S* on the LHS.



Parsing Challenges

- Recap: Parsing-as-Search
- **Parsing Challenges**
  - **Ambiguity**
  - Repeated Substructure
  - Recursion
- Strategy: Dynamic Programming
- Grammar Equivalence
- CKY parsing algorithm

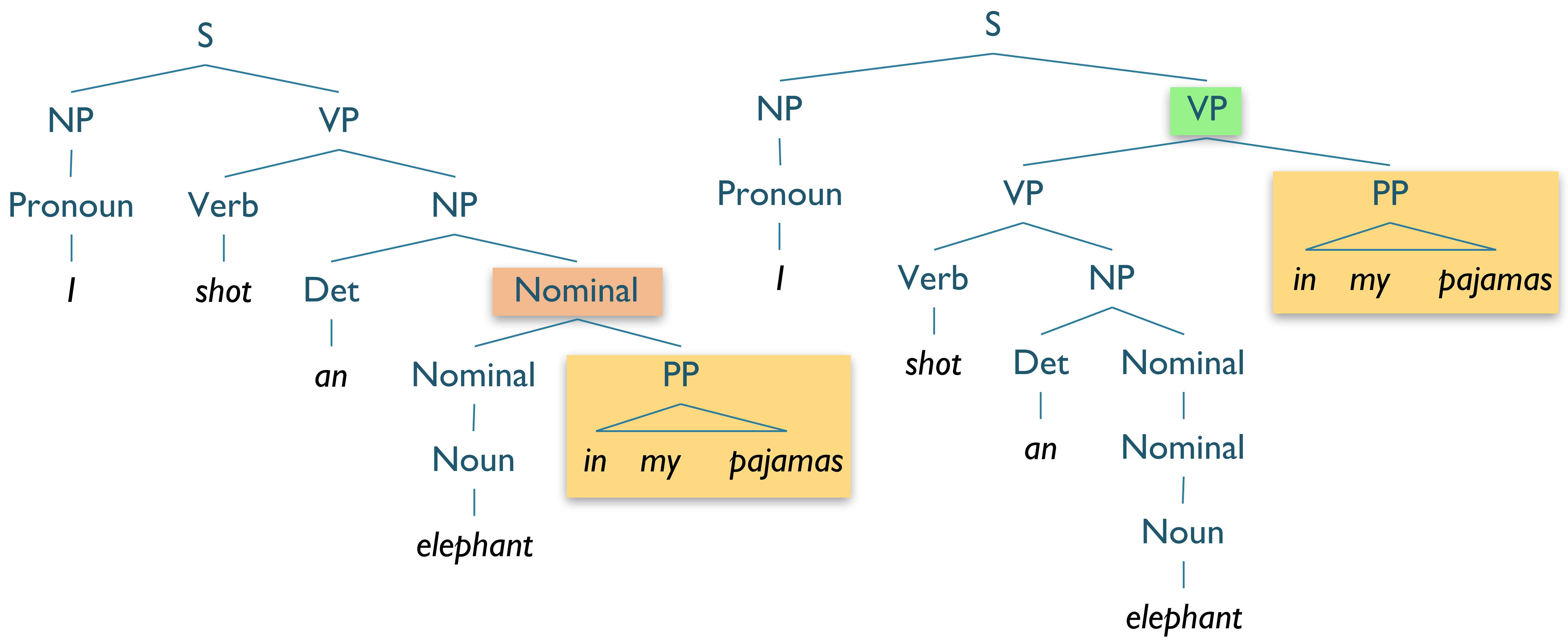
Parsing Ambiguity

- **Lexical Ambiguity**:
  - Book/NN → *I left a **book** on the table.*
  - Book/VB → ***Book** that flight.*
- Structural Ambiguity

Attachment Ambiguity

"One morning, I shot an elephant in my pajamas.
How he got into my pajamas, I'll never know." — *Groucho Marx*

Attachment Ambiguity

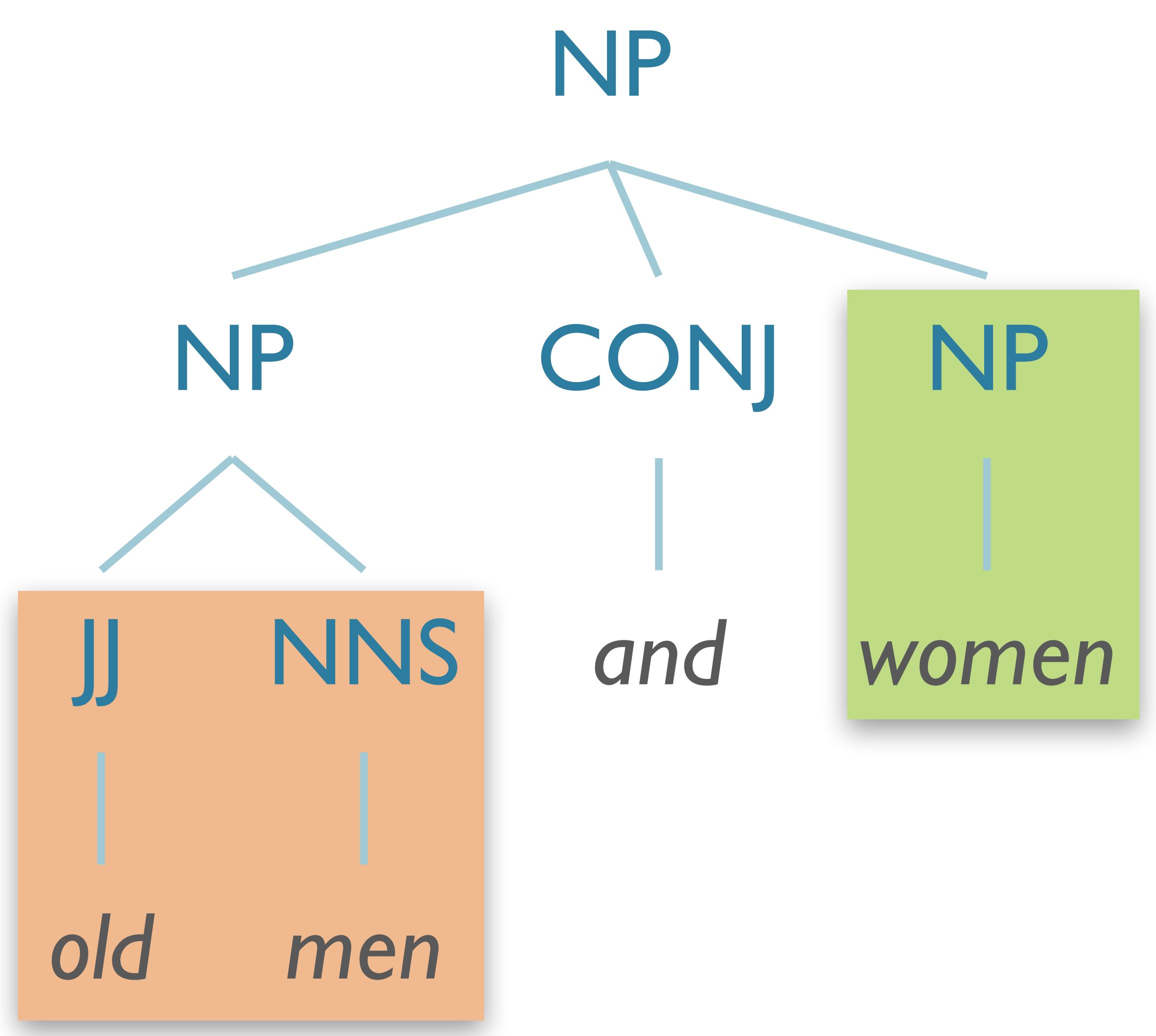


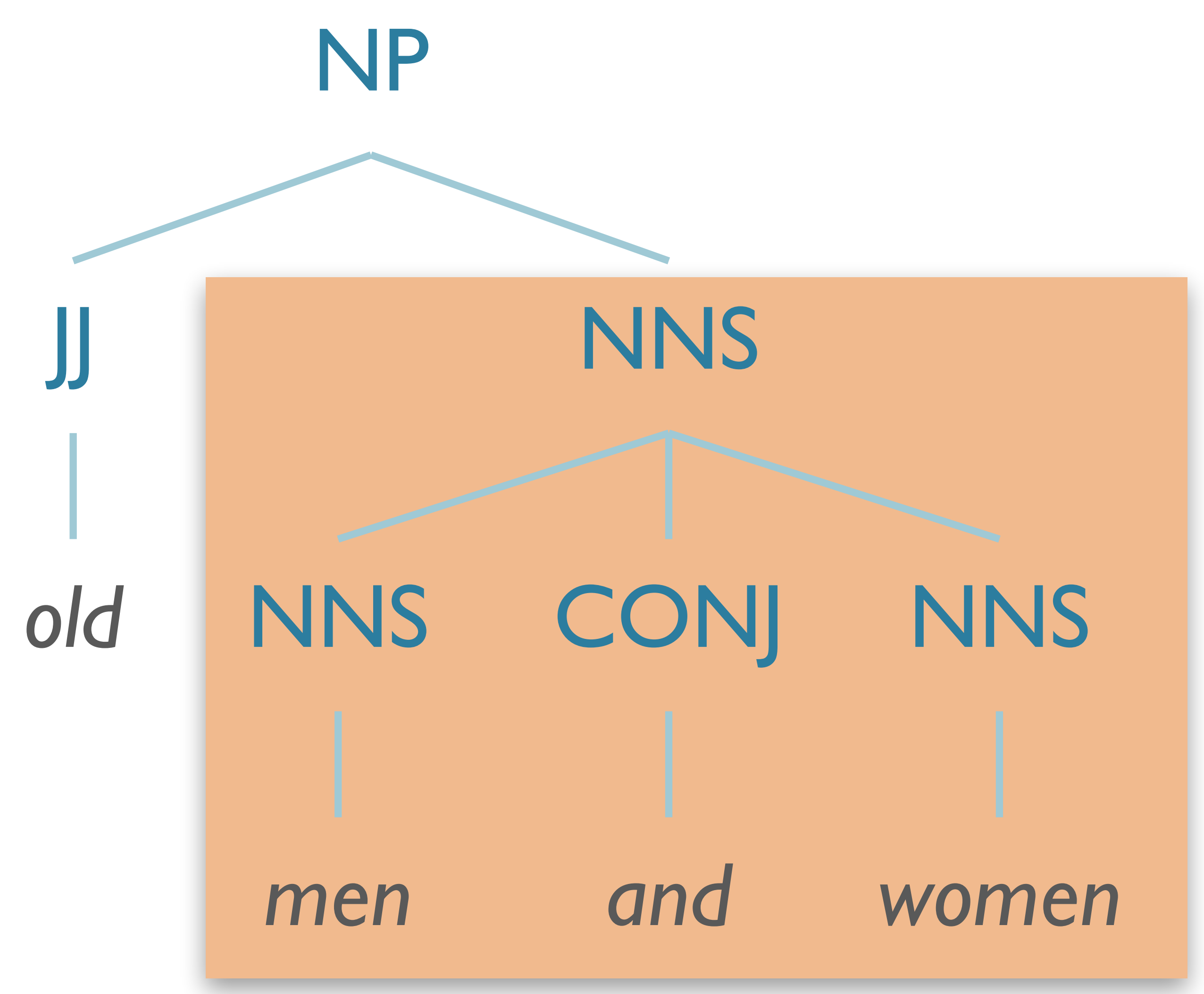
“We saw the Eiffel Tower flying to Paris”

Coordination Ambiguity:

[old men] *and* ***[women]***

[old ***[men and women]****]*





Local vs. Global Ambiguity

- *Local* ambiguity:
  - Ambiguity that cannot contribute to a full, valid parse
  - e.g. *Book/NN* in *“Book that flight”*
- *Global* ambiguity
  - Multiple valid parses

Why is Ambiguity a Problem?

- *Local* ambiguity:
  - increased processing time

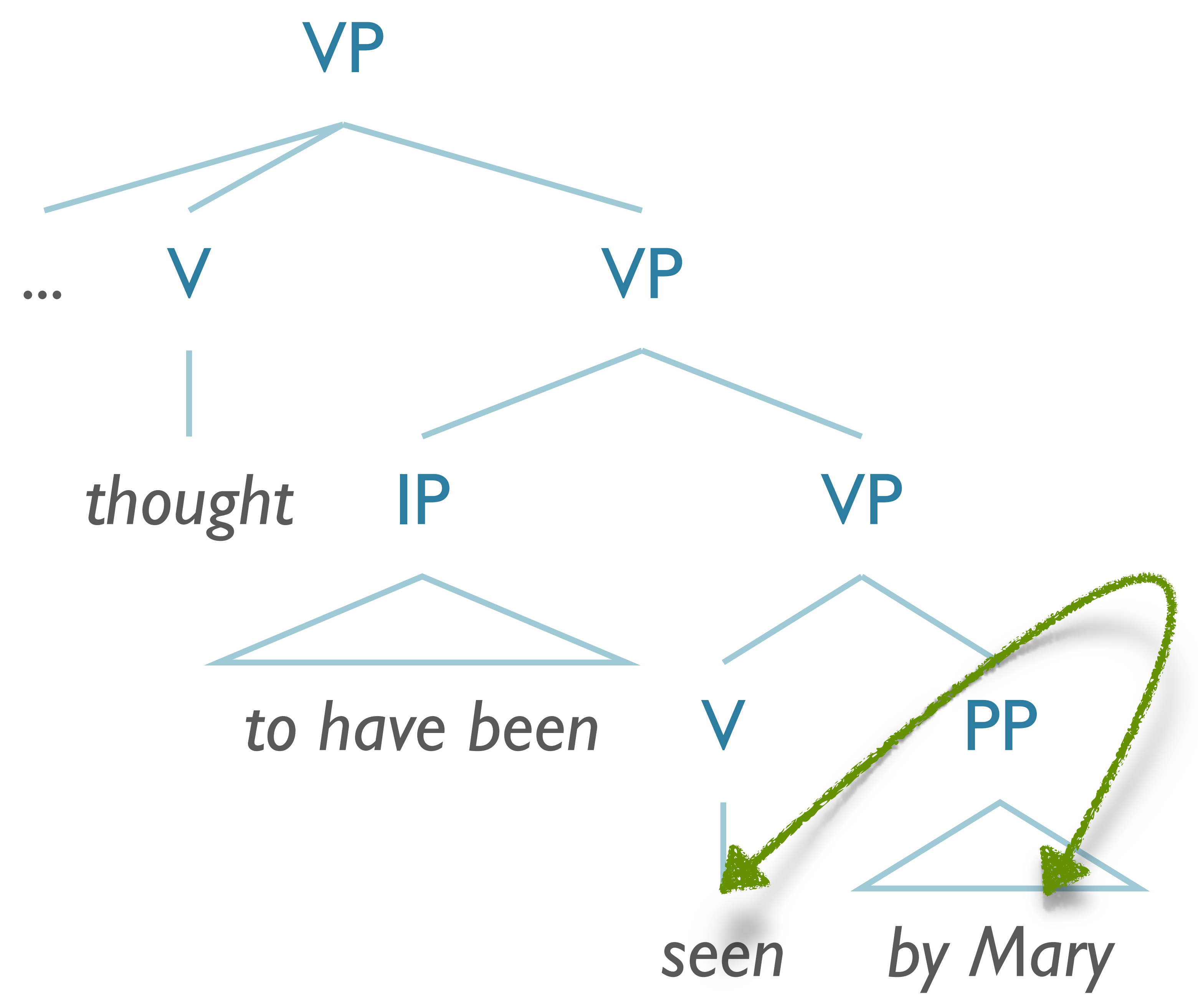
- *Global* ambiguity:
  - Would like to yield only "reasonable" parses
  - Ideally, the one that was intended*

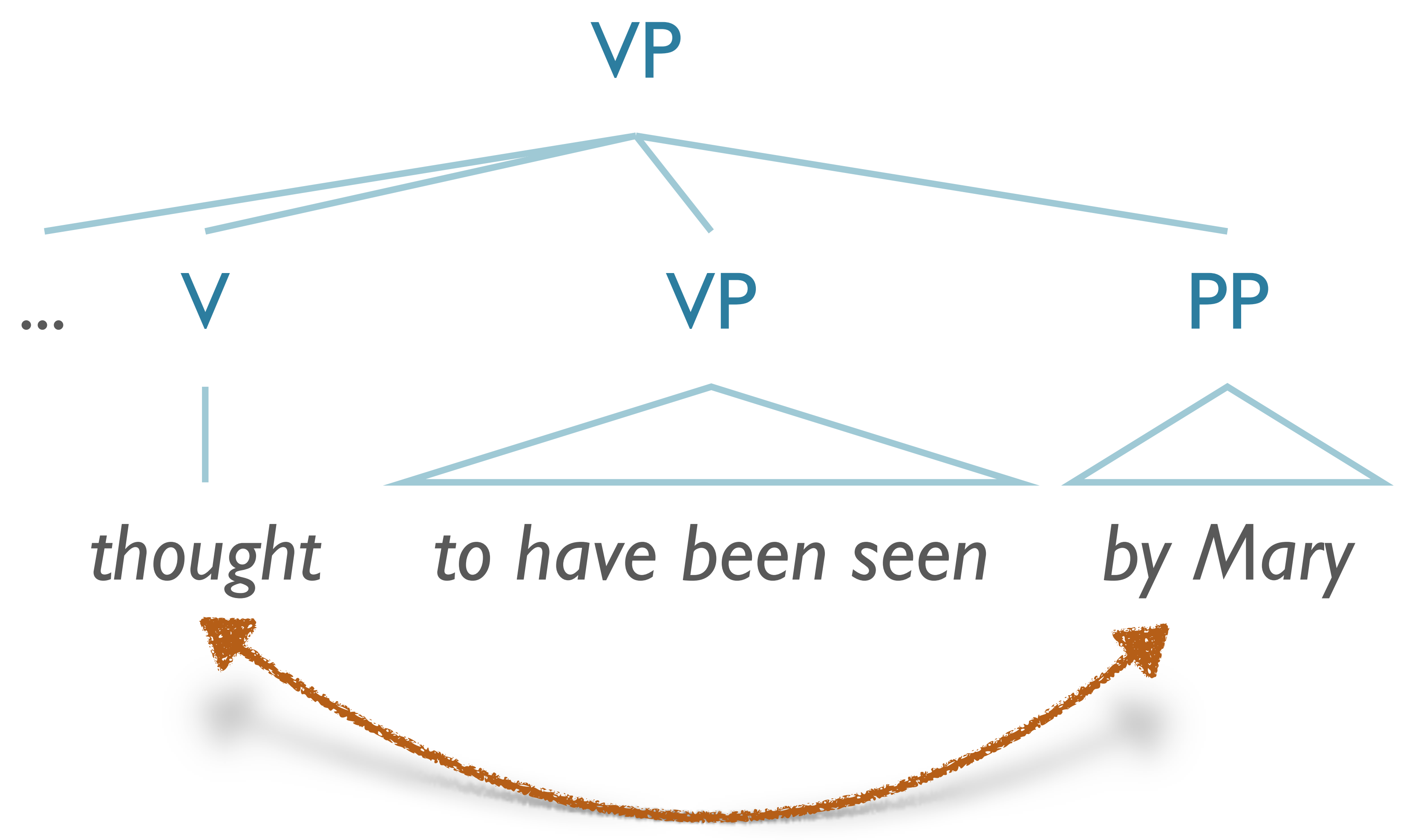
Solution to Ambiguity?

- ***Dis*ambiguation!**
- Different possible strategies to select correct interpretation:

Disambiguation Strategy: Statistical

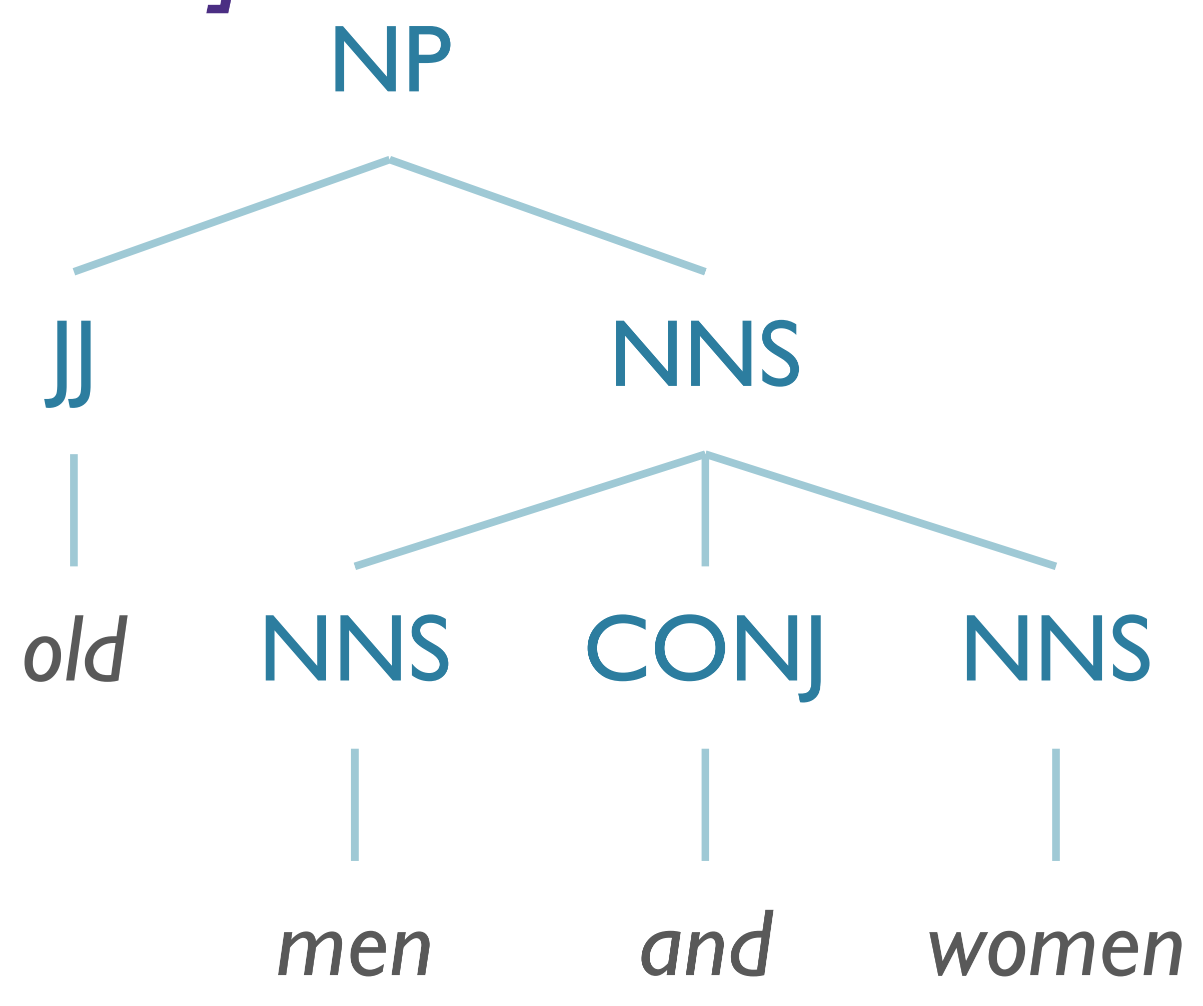
- Some prepositional structs more likely to attach high/low
  - *John was thought to have been seen by Mary*
    - Mary could be doing the seeing or thinking — seeing more likely

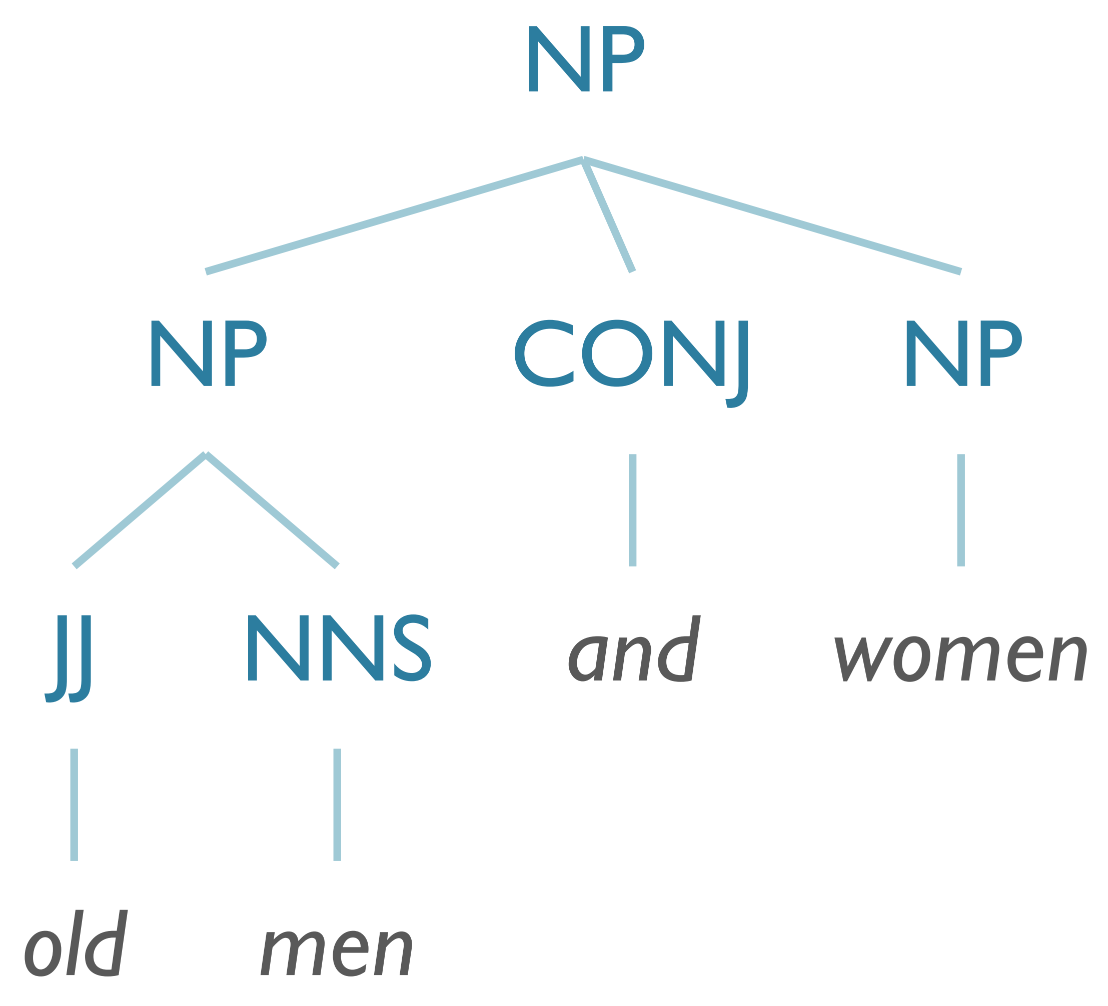




Disambiguation Strategy: Statistical

- Some phrases more likely overall
  - *[old [men and women]] is a more common construction than [old men] and [women]*





Disambiguation Strategy: Semantic

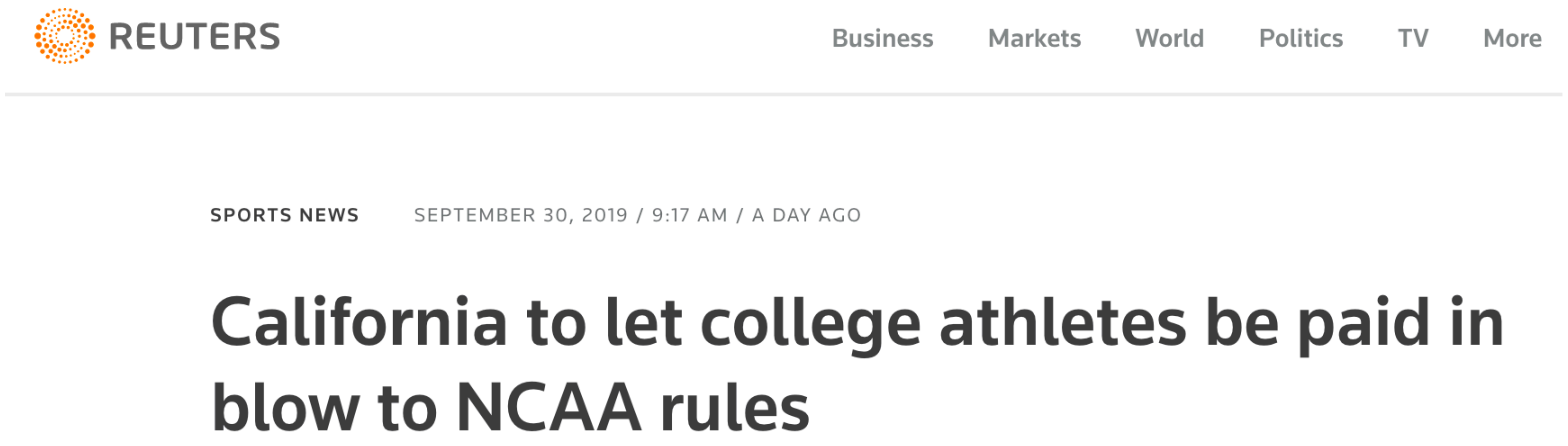
- Some interpretations we know to be semantically impossible
  - *Eiffel tower* as subject of *fly*

Disambiguation Strategy: Pragmatic

- Some interpretations are possible, unlikely given world knowledge
  - e.g. elephants and pajamas

Incremental Parsing and Garden Paths

- Idea: model *left-to-right* nature of (English) text
- Problem: “garden path” sentences



SPORTS NEWS SEPTEMBER 30, 2019 / 9:17 AM / A DAY AGO

California to let college athletes be paid in blow to NCAA rules

https://www.reuters.com/article/us-sport-california-education/california-to-let-college-athletes-be-paid-in-blow-to-ncaa-rules-idUSKBN1WF1SR

Disambiguation Strategy:

🤷‍♂️

- Alternatively, keep all parses
  - *(Might even be the appropriate action for some jokes)*

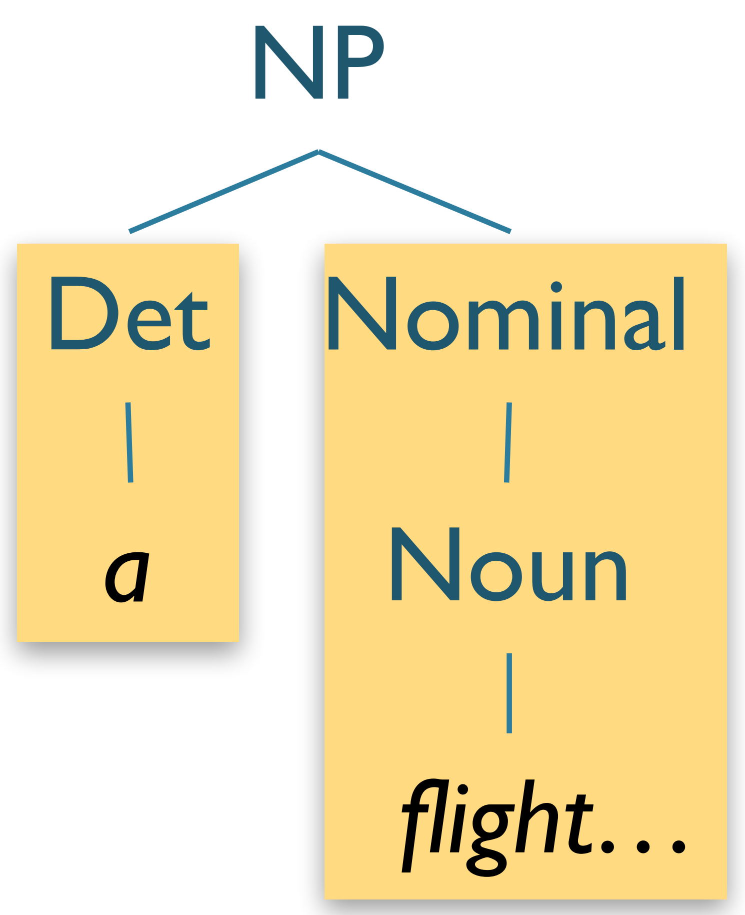
Parsing Challenges

- Recap: Parsing-as-Search
- **Parsing Challenges**
  - Ambiguity
  - **Repeated Substructure**
  - Recursion
- Strategy: Dynamic Programming
- Grammar Equivalence
- CKY parsing algorithm

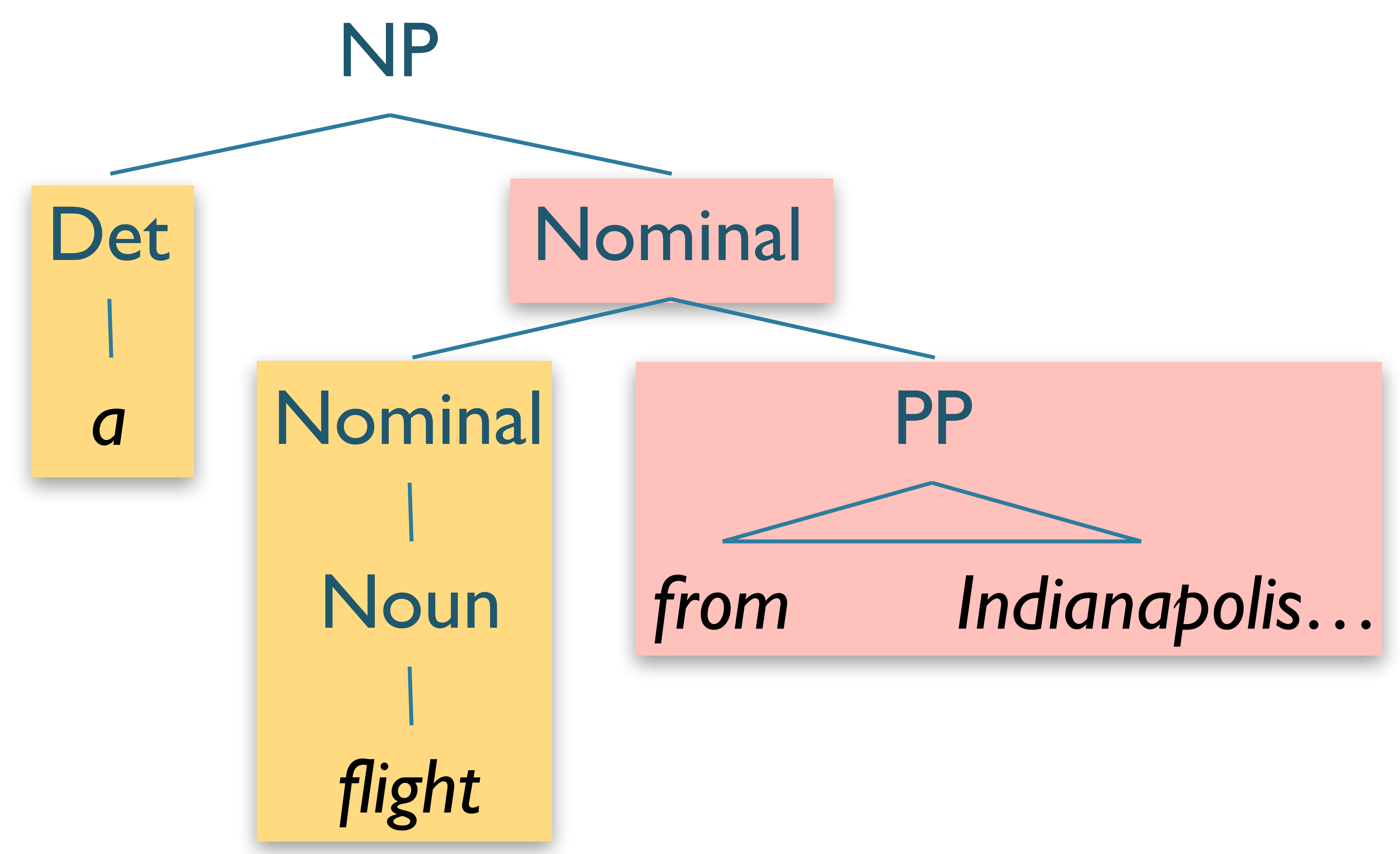
Repeated Work

- Search (top-down/bottom-up) both lead to repeated substructures
  - Globally bad parses can construct good subtrees
  - …will reconstruct along another branch
  - No static backtracking can avoid
- Efficient parsing techniques require storage of partial solutions
- Example: *a flight from Indianapolis to Houston on TWA*

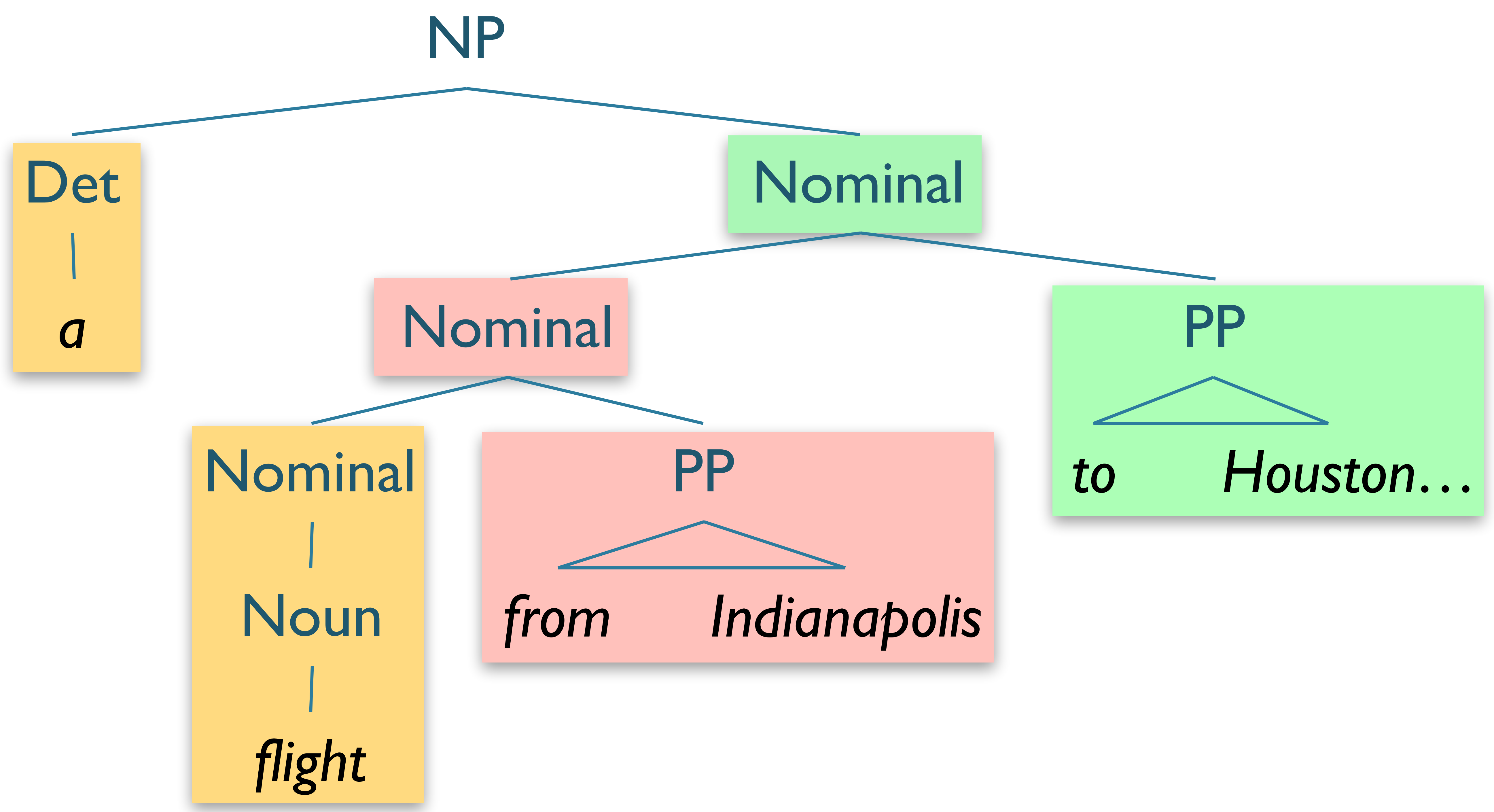
Shared Sub-Problems



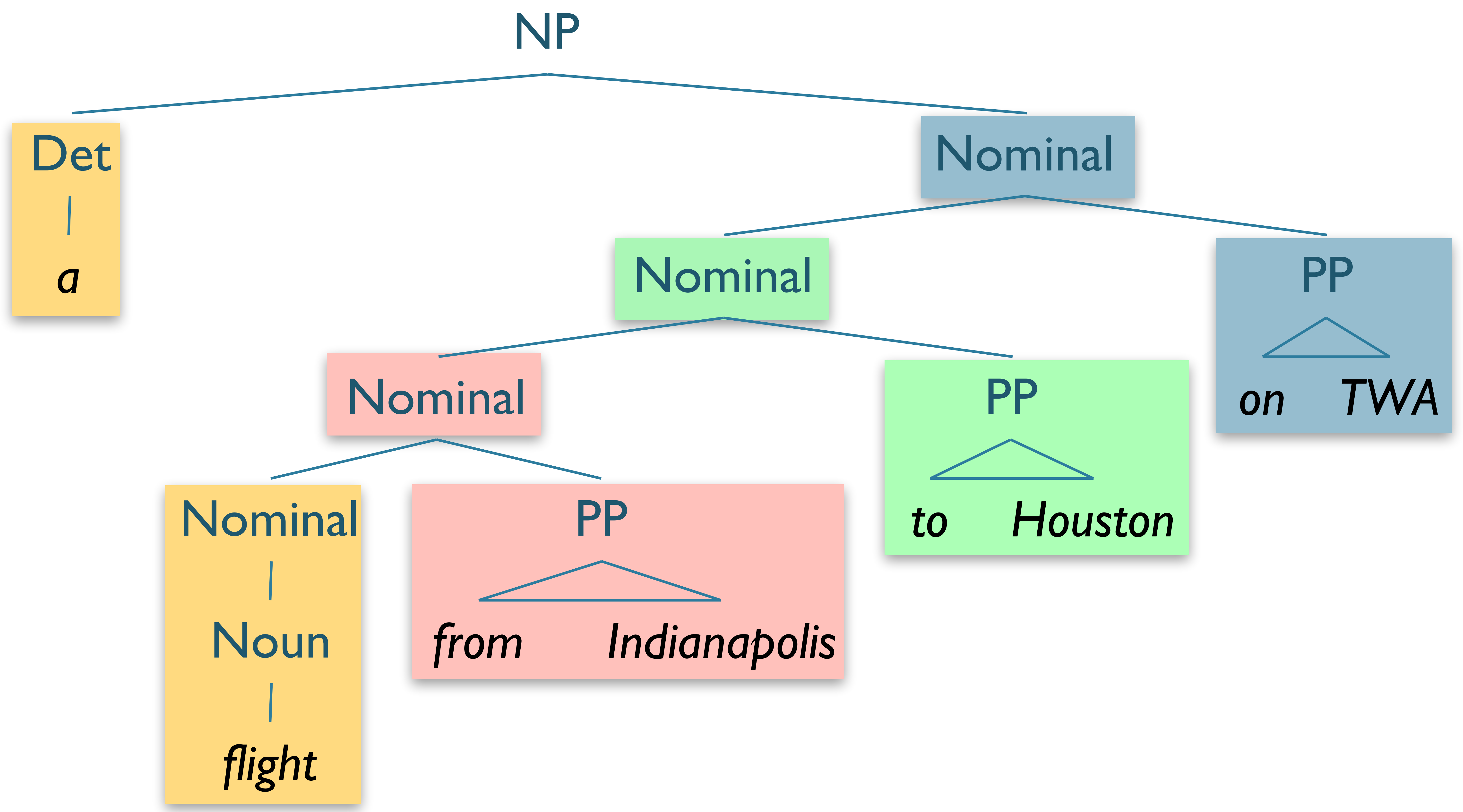
Shared Sub-Problems



Shared Sub-Problems



Shared Sub-Problems

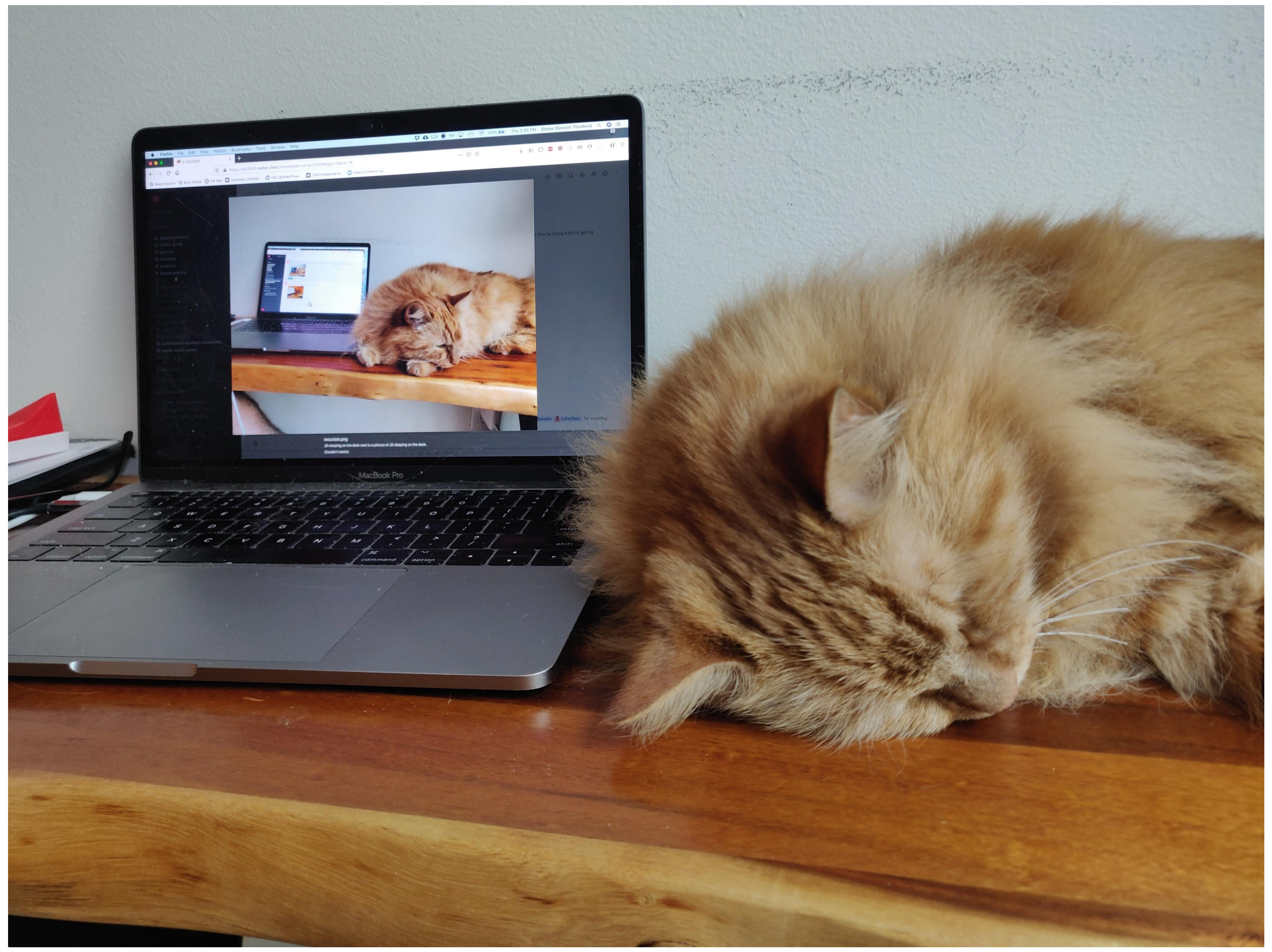


Parsing Challenges

- Recap: Parsing-as-Search
- **Parsing Challenges**
  - Ambiguity
  - Repeated Substructure
  - **Recursion**
- Strategy: Dynamic Programming
- Grammar Equivalence
- CKY parsing algorithm

Recursion

- Many grammars have recursive rules
  - ***S*** → ***S*** *Conj* ***S***
- In search approaches, recursion is problematic
  - Can yield infinite searches
  - Top-down especially vulnerable



Roadmap

- Recap: Parsing-as-Search
- Parsing Challenges
- **Strategy: Dynamic Programming**
- Grammar Equivalence
- CKY parsing algorithm

Dynamic Programming

- Challenge:
  - Repeated substructure → Repeated Work
- Insight:
  - Global parse composed of sub-parses
  - Can record these sub-parses and re-use
- Dynamic programming avoids repeated work by recording the subproblems
  - Here, stores subtrees

Parsing with Dynamic Programming

- Avoids repeated work
- Allows implementation of (relatively) efficient parsing algorithms
    - Polynomial time in input length
    - Typically cubic (***n***^{3}) or less
- Several different implementations
    - Cocke-Kasami-Younger (CKY) algorithm
    - Earley algorithm
    - Chart parsing

Roadmap

- Recap: Parsing-as-Search
- Parsing Challenges
- Strategy: Dynamic Programming
- **Grammar Equivalence**
- CKY parsing algorithm

Grammar Equivalence and Form

- *Weak* Equivalence
  - **Accepts** same language
  - May produce **different** structures

- *Strong* Equivalence
  - Accepts same language
  - Produces **same** structures

Grammar Equivalence and Form

- Reason?
  - We can create a weakly-equivalent grammar that allows for greater efficiency
  - This is required by the CKY algorithm

Chomsky Normal Form (CNF)

- Required by CKY Algorithm
- All productions are of the form:
  - $A \rightarrow B\ C$
  - $A \rightarrow$ **a**
- Most of our grammars are not of this form:
  - $S \rightarrow$ *Wh-NP Aux NP VP*
- Need a general conversion procedure

CNF Conversion

Hybrid productions:

INF-VP → **to** *VP*

Unit productions:

A → *B*

Long productions:

A → *B C D …*

CNF Conversion: Hybrid Productions

- Hybrid production:
  - Replace all terminals with dummy non-terminal
  - *INF-VP* → **to** *VP*
    - *INF-VP* → *TO VP*
    - *TO* → **to**

CNF Conversion: Unit Productions

- Unit productions:
  - Rewrite RHS with RHS of all derivable, non-unit productions
  - If $A \overset{*}{\Rightarrow} B$ and $B \rightarrow$ **w, add** $A \rightarrow$ **w**
  - **[**$A \overset{*}{\Rightarrow}$ *B: B is reachable from A by a sequence of unit productions]*
- *Nominal* → *Noun*, *Noun* → **dog**
  - *Nominal* → **dog**
  - *Noun* → **dog**

CNF Conversion: Long Productions

- Long productions

S → *Aux NP VP*	
S → ***X1*** *VP*	***X1*** → *Aux NP*

- Introduce unique nonterminals, and spread over rules

CNF Conversion

Convert terminals in hybrid rules to dummy non-terminals

Convert unit productions

Binarize long production rules

$\mathscr{L}_1$ Grammar	$\mathscr{L}_1$ in CNF
S → NP VP	*S → NP VP*
S → Aux NP VP	*S → X1 VP*
	X1 → Aux NP
S → VP	*S → book I include I prefer*
	S → Verb NP
	S → X2 PP
	S → Verb PP
	S → VP PP
NP → Pronoun	*NP → I I she I me*
NP → Proper-Noun	*NP → TWA I Houston*
NP → Det Nominal	*NP → Det Nominal*
Nominal → Noun	*Nominal → book I flight I meal I money*
Nominal → Nominal Noun	*Nominal → Nominal Noun*
Nominal → Nominal PP	*Nominal → Nominal PP*
VP → Verb	*VP → book I include I prefer*
VP → Verb NP	*VP → Verb NP*
VP → Verb NP PP	*VP → X2 PP*
	X2 → Verb NP
VP → Verb PP	*VP → Verb PP*
VP → VP PP	*VP → VP PP*
PP → Preposition NP	*PP → Preposition NP*

$\mathscr{L}_1$ Grammar	$\mathscr{L}_1$ in 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
	S → Verb PP
	S → VP PP
NP → Pronoun	*NP → I | she | me*
NP → Proper-Noun	*NP → TWA | Houston*
NP → Det Nominal	*NP → Det Nominal*
Nominal → Noun	*Nominal → book | flight | meal | money*
Nominal → Nominal Noun	*Nominal → Nominal Noun*
Nominal → Nominal PP	*Nominal → Nominal PP*
VP → Verb	*VP → book | include | prefer*
VP → Verb NP	*VP → Verb NP*
VP → Verb NP PP	*VP → X2 PP*
	X2 → Verb NP
VP → Verb PP	*VP → Verb PP*
VP → VP PP	*VP → VP PP*
PP → Preposition NP	*PP → Preposition NP*

$\mathscr{L}_1$ Grammar	$\mathscr{L}_1$ in CNF
S → NP VP	*S → NP VP*
S → Aux NP VP	*S → X1 VP*
	X1 → Aux NP
S → VP	*S → book I include I prefer*
	S → Verb NP
	S → X2 PP
	S → Verb PP
	S → VP PP
NP → Pronoun	*NP → I I she I me*
NP → Proper-Noun	*NP → TWA I Houston*
NP → Det Nominal	*NP → Det Nominal*
Nominal → Noun	*Nominal → book I flight I meal I money*
Nominal → Nominal Noun	*Nominal → Nominal Noun*
Nominal → Nominal PP	*Nominal → Nominal PP*
VP → Verb	*VP → book I include I prefer*
VP → Verb NP	*VP → Verb NP*
VP → Verb NP PP	*VP → X2 PP*
	X2 → Verb NP
VP → Verb PP	*VP → Verb PP*
VP → VP PP	*VP → VP PP*
PP → Preposition NP	*PP → Preposition NP*

Roadmap

- Recap: Parsing-as-Search
- Parsing Challenges
- Strategy: Dynamic Programming
- Grammar Equivalence
- **CKY parsing algorithm**

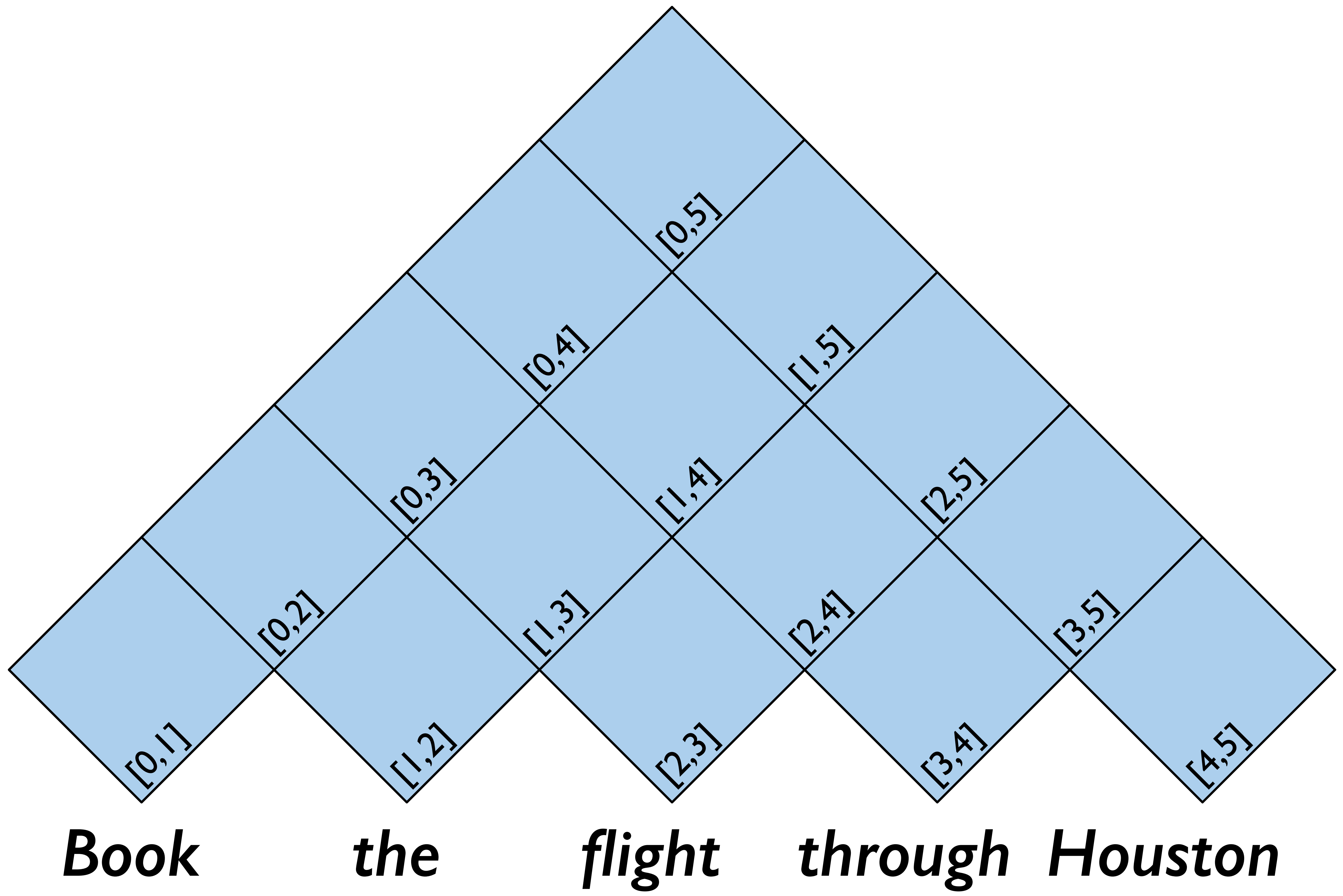
CKY Parsing

- (Relatively) efficient parsing algorithm
- Based on tabulating substring parses to avoid repeat work
- Approach:
  - Use CNF Grammar
  - Build an $(n + 1) \times (n + 1)$ matrix to store subtrees
    - Upper triangular portion
  - Incrementally build parse spanning whole input string

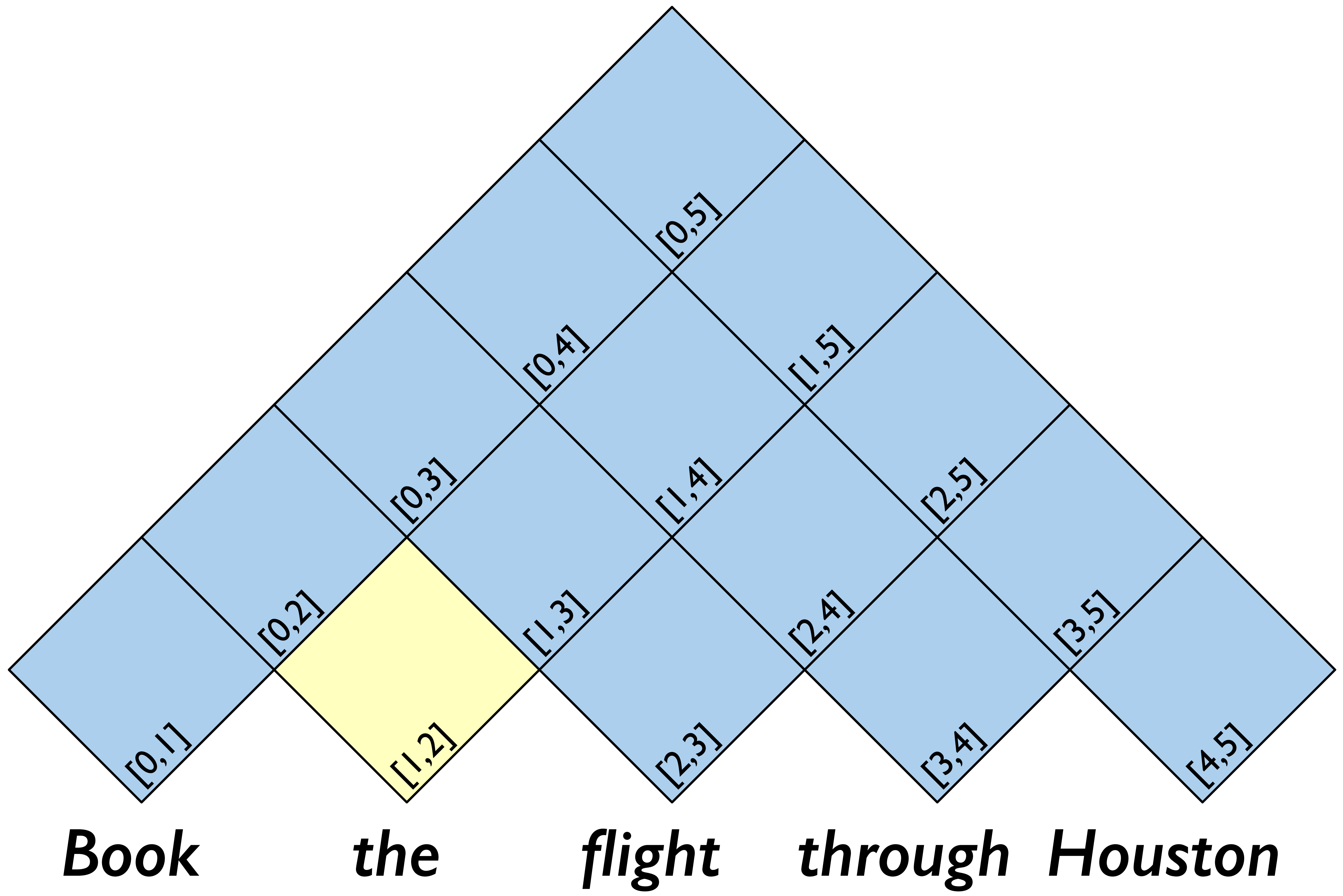
CKY Matrix

Book	*the*	*flight*	*through*	*Houston*
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

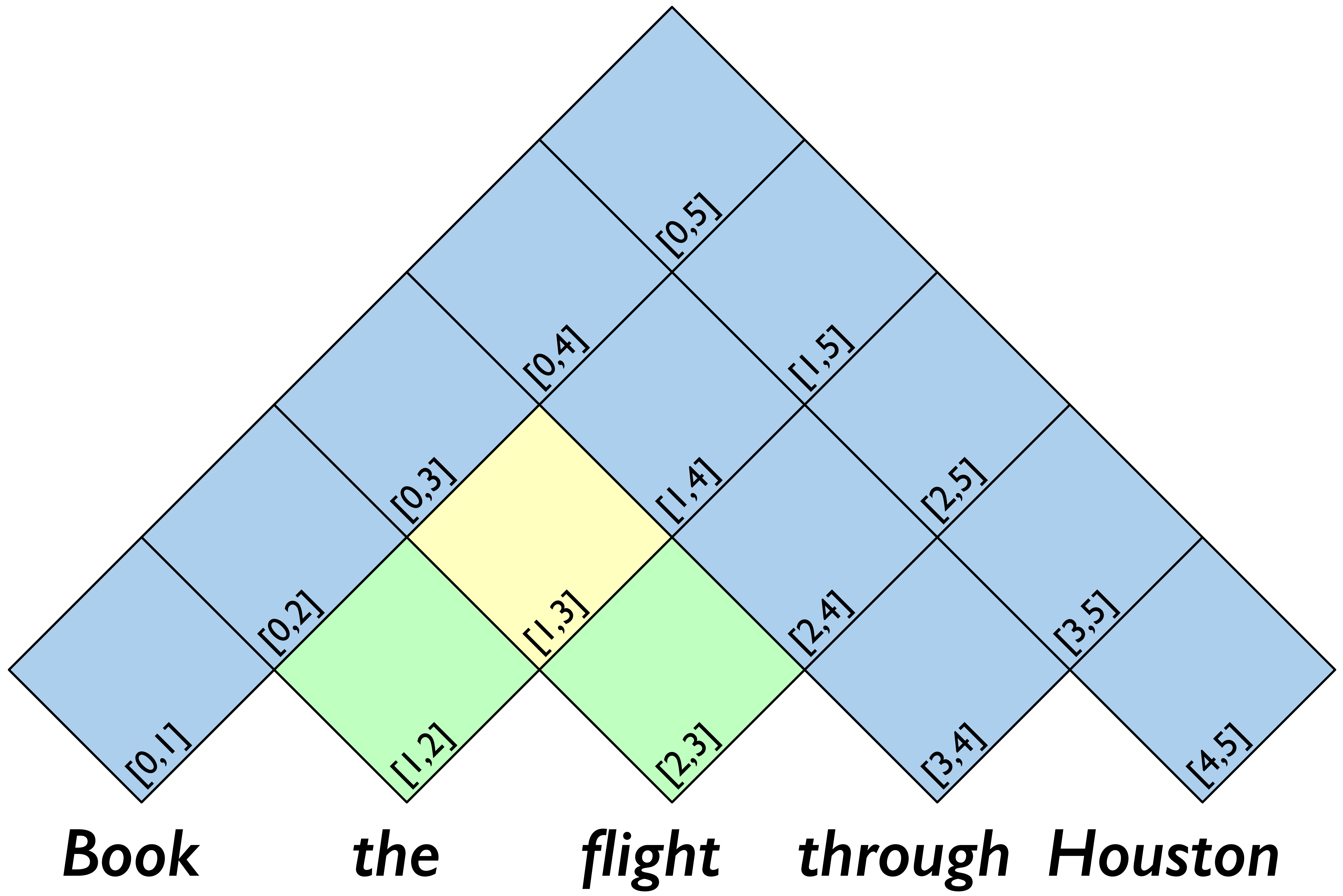
CKY Matrix


[0,5]
[0,4]
[1,5]
[0,3]
[1,4]
[2,5]
[0,2]
[1,3]
[2,4]
[3,5]
[0,1]
[1,2]
[2,3]
[3,4]
[4,5]
Book
the
flight
through
Houston

CKY Matrix


[0,5]
[0,4]
[1,5]
[0,3]
[1,4]
[2,5]
[0,2]
[1,3]
[2,4]
[3,5]
[0,1]
[1,2]
[2,3]
[3,4]
[4,5]
Book
the
flight
through
Houston

CKY Matrix


[0,5]
[0,4]
[1,5]
[0,3]
[1,4]
[2,5]
[0,2]
[1,3]
[2,4]
[3,5]
[0,1]
[1,2]
[2,3]
[3,4]
[4,5]
Book
the
flight
through
Houston

Dynamic Programming in CKY

- Key idea:
  - for $\boldsymbol{i < k < j}$
  - …and a parse spanning substring [$\boldsymbol{i}$, $\boldsymbol{j}$]
  - There is a $\boldsymbol{k}$ such that there are parses spanning [$\boldsymbol{i}$, $\boldsymbol{k}$] and [$\boldsymbol{k}$, $\boldsymbol{j}$]
  - We can construct parses for whole sentences by building from these partial parses
- So to have a rule $\boldsymbol{A \to B\ C}$ in [$\boldsymbol{i}$, $\boldsymbol{j}$]
  - Must have $\boldsymbol{B}$ in [$\boldsymbol{i}$, $\boldsymbol{k}$] and $\boldsymbol{C}$ in [$\boldsymbol{k}$, $\boldsymbol{j}$] for some $\boldsymbol{i < k < j}$
  - CNF forces this for all $\boldsymbol{j > i + 1}$

HW #2

LING 571
Deep Processing Techniques for NLP
October 7, 2020

Goals

- Begin development of CKY parser

- First stage: Conversion to CNF
  - Develop Representation for CFG
  - Manipulate/Transform Grammars
  - Investigate weakly equivalent grammars

Task

- Conversion:
  - Read in grammar rules from arbitrary CFG
  - Convert to CNF
  - Write out new grammar
- Validation:
  - Parse test sentences with original CFG
  - Parse test sentences with CFG in CNF

Approach

- May use any programming language
  - In keeping with course policies
- May use existing models/packages to represent rules
  - Need RULE, RHS, LHS, etc
  - NLTK, Stanford
- ***Conversion code must be your own***

Data

- ATIS (Air Travel Information System) data
  - Grammar provided in nltk-data
  - Terminals in double-quotes
    - *the* → “the”
  - All required files on patas dropbox
- **NOTE**:
  - Grammar is fairly large (~193K Productions)
  - Grammar is fairly ambiguous (Test sentences may have 100 parses)
  - You will likely want to develop against a smaller grammar
  - You must submit a *condor* .cmd file

NLTK Grammars

```
>>> gr1 = nltk.data.load('grammars/large_grammars/
atis.cfg')

>>> gr1.productions()[0]
ABBCL_NP -> QUANP_DTI QUANP_DTI QUANP_CD AJP_JJ NOUN_NP
PRPRTCL_VBG

>>> gr1.productions()[0].lhs()
ABBCL_NP

>>> gr1.productions(lhs=gr1.productions()[1].lhs())
[ADJ_ABL -> only, ADJ_ABL->such]
```