PCFGs:
Parsing & Evaluation

LING 571 — Deep Processing Techniques for NLP
October 9, 2019
Shane Steinert-Threlkeld

Roadmap

- Recap:
  - CKY + back-pointers
  - PCFGs
- PCFG Parsing (PCKY)
- Inducing a PCFG
- Evaluation
- [Earley parsing]
- HW3 + collaboration

CKY Follow-up: Backpointers

Backpointers

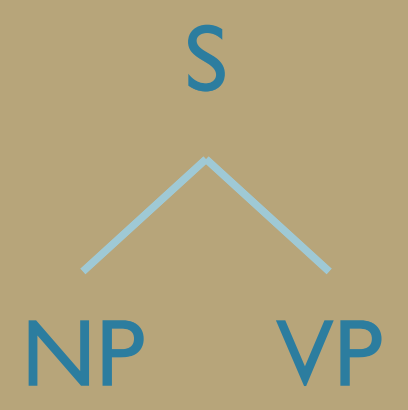
- Instead of list of possible nonterminals for that node, each cell should have:
  - Nonterminal for the node
  - Pointer to left and right children cells
    - Either direct pointer to cell, or indices

For example:

```
bp_2 = BackPointer()
bp_2.l_child = [X2, (1,4)]
bp_2.r_child = [PP, (4,6)]
```

```
cky_table[0,6][S] = {(NP, (0,1)),
                     VP, (1,6))}
```

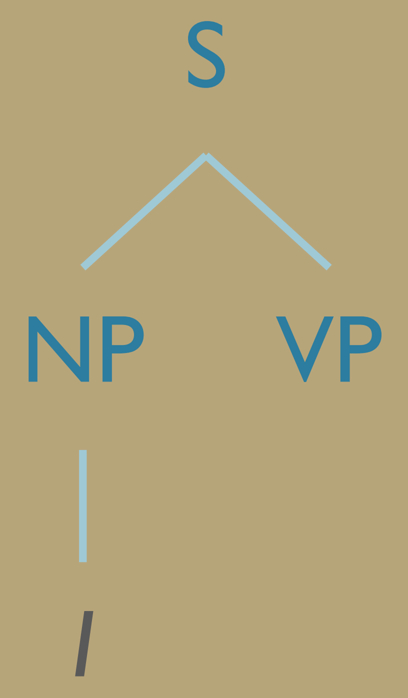
NP, Pronoun [0,1]	**S** [0,2]	[0,3]	**S** [0,4]	[0,5]	**S** [0,6]
	Verb, VP, S [1,2]	[1,3]	**VP, X2, S** [1,4]	[1,5]	**VP, X2, S** [1,6]
		Det [2,3]	**NP** [2,4]	[2,5]	**NP** [2,6]
			Noun, Nom [3,4]	[3,5]	**Nom** [3,6]
				Prep [4,5]	**PP** [4,6]
					NNP, NP [5,6]



I *prefer* *a* *flight* *on* *TWA*

```
cky_table[0,6][S] = {(NP, (0,1),
                      VP, (1,6))}
  cky_table[0,1][NP] = {('I')}
```

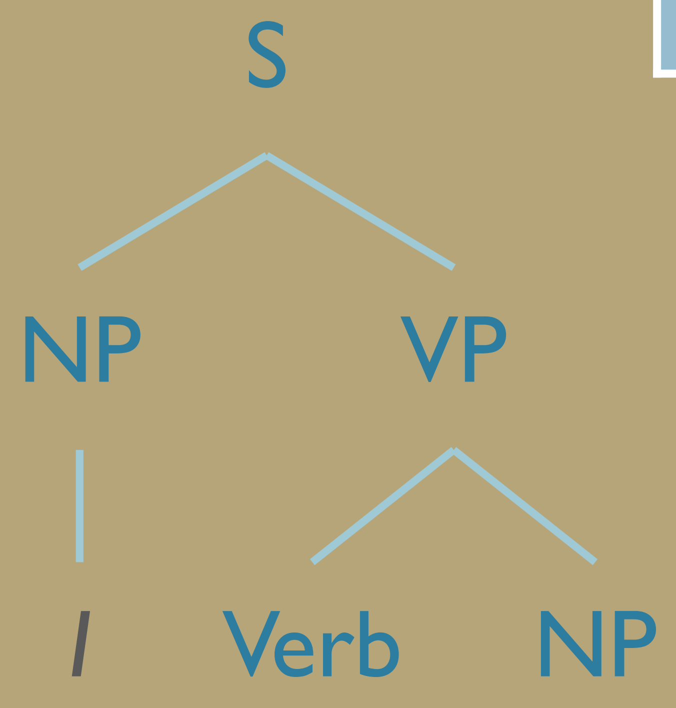
NP, Pronoun [0,1]	S [0,2]	[0,3]	S [0,4]	[0,5]	S [0,6]
	Verb, VP, S [1,2]	[1,3]	VP, X2, S [1,4]	[1,5]	VP, X2, S [1,6]
		Det [2,3]	NP [2,4]	[2,5]	NP [2,6]
			Noun, Nom [3,4]	[3,5]	Nom [3,6]
				Prep [4,5]	PP [4,6]
					NNP, NP [5,6]



I prefer a flight on TWA

```
cky_table[0,6][S] = {(NP, (0,1)),
                      VP, (1,6))}
   cky_table[0,1][NP] = {('I')}
cky_table[1,6][VP] = {(Verb, (1,2),
                       NP, (2,6)),
                      (X2, (1,4),
                       PP, (4,6))}
```

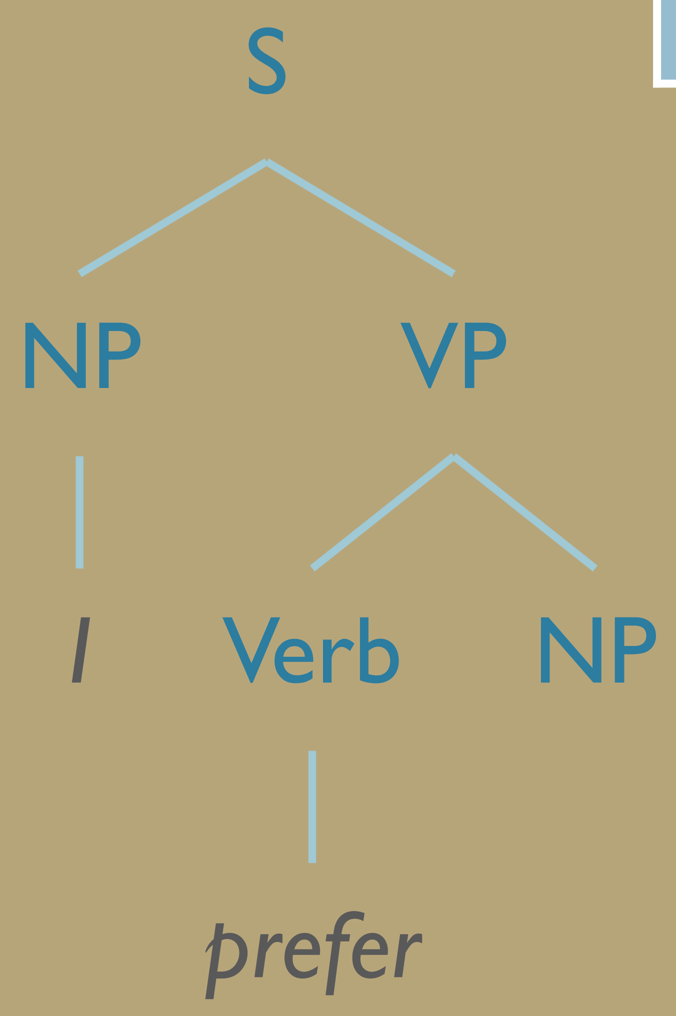
NP, Pronoun [0,1]	S [0,2]	[0,3]	S [0,4]	[0,5]	S [0,6]
	Verb, VP, S [1,2]	[1,3]	VP, X2, S [1,4]	[1,5]	VP, X2, S [1,6]
		Det [2,3]	NP [2,4]	[2,5]	NP [2,6]
			Noun, Nom [3,4]	[3,5]	Nom [3,6]
				Prep [4,5]	PP [4,6]
					NNP, NP [5,6]



I prefer a flight on TWA

```
cky_table[0,6][S] = {(NP, (0,1),
                      VP, (1,6))}
  cky_table[0,1][NP] = {('I')}
cky_table[1,6][VP] = {(Verb, (1,2),
                       NP, (2,6)),
                      (X2, (1,4),
                       PP, (4,6))}
cky_table[1,2][Verb] = {('prefer')}
```

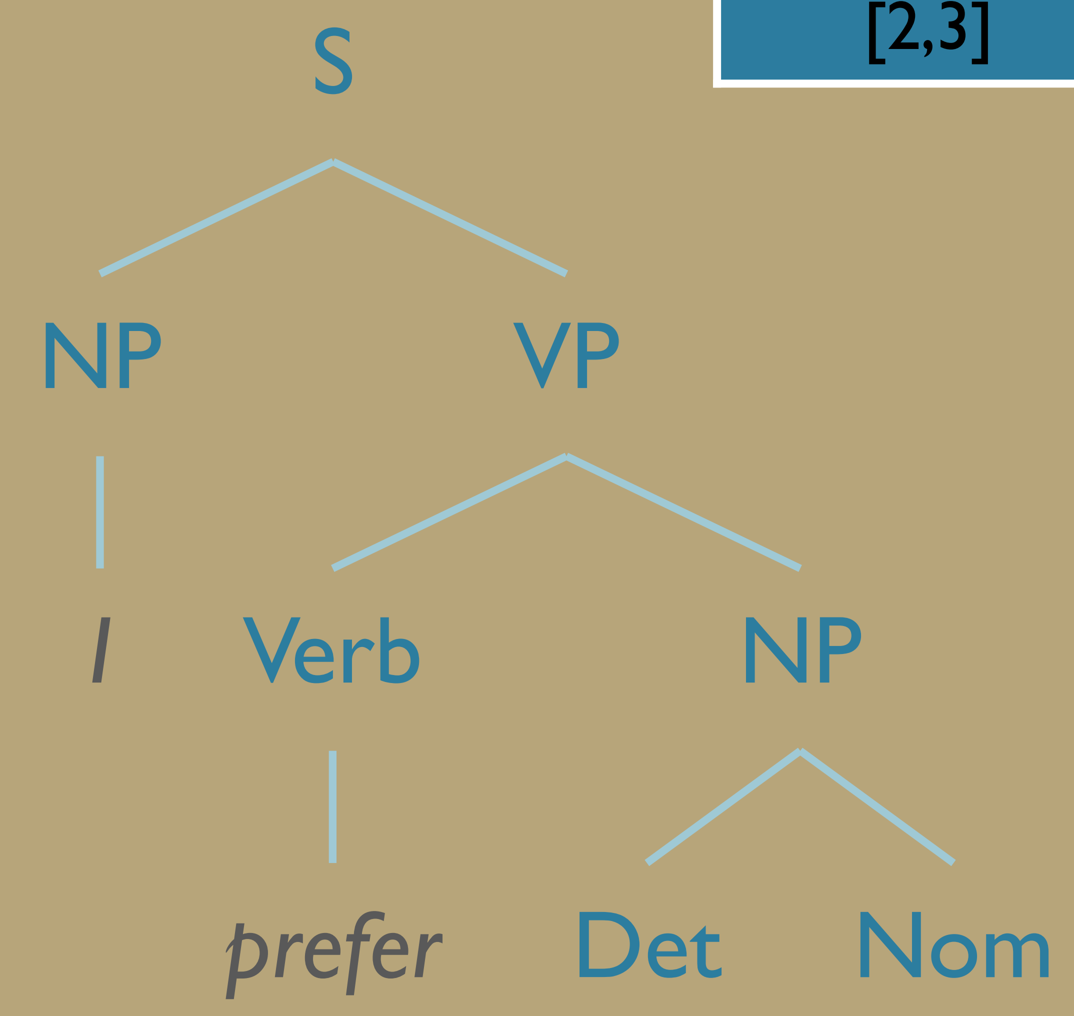
NP, Pronoun [0,1]	**S** [0,2]	[0,3]	**S** [0,4]	[0,5]	**S** [0,6]
	Verb, VP, S [1,2]	[1,3]	**VP, X2, S** [1,4]	[1,5]	**VP, X2, S** [1,6]
		Det [2,3]	**NP** [2,4]	[2,5]	**NP** [2,6]
			Noun, Nom [3,4]	[3,5]	**Nom** [3,6]
				Prep [4,5]	**PP** [4,6]
					NNP, NP [5,6]



I *prefer* *a* *flight* *on* *TWA*

```
 cky_table[0,6][S] = {(NP, (0,1),
                      VP, (1,6))}
   cky_table[0,1][NP] = {('I')}
cky_table[1,6][VP] = {(Verb, (1,2),
                       NP, (2,6)),
                      (X2, (1,4),
                       PP, (4,6))}
cky_table[1,2][Verb] = {('prefer')}
 cky_table[2,6][NP] = {(Det, (2,3),
                        Nom, (3,6))}
```

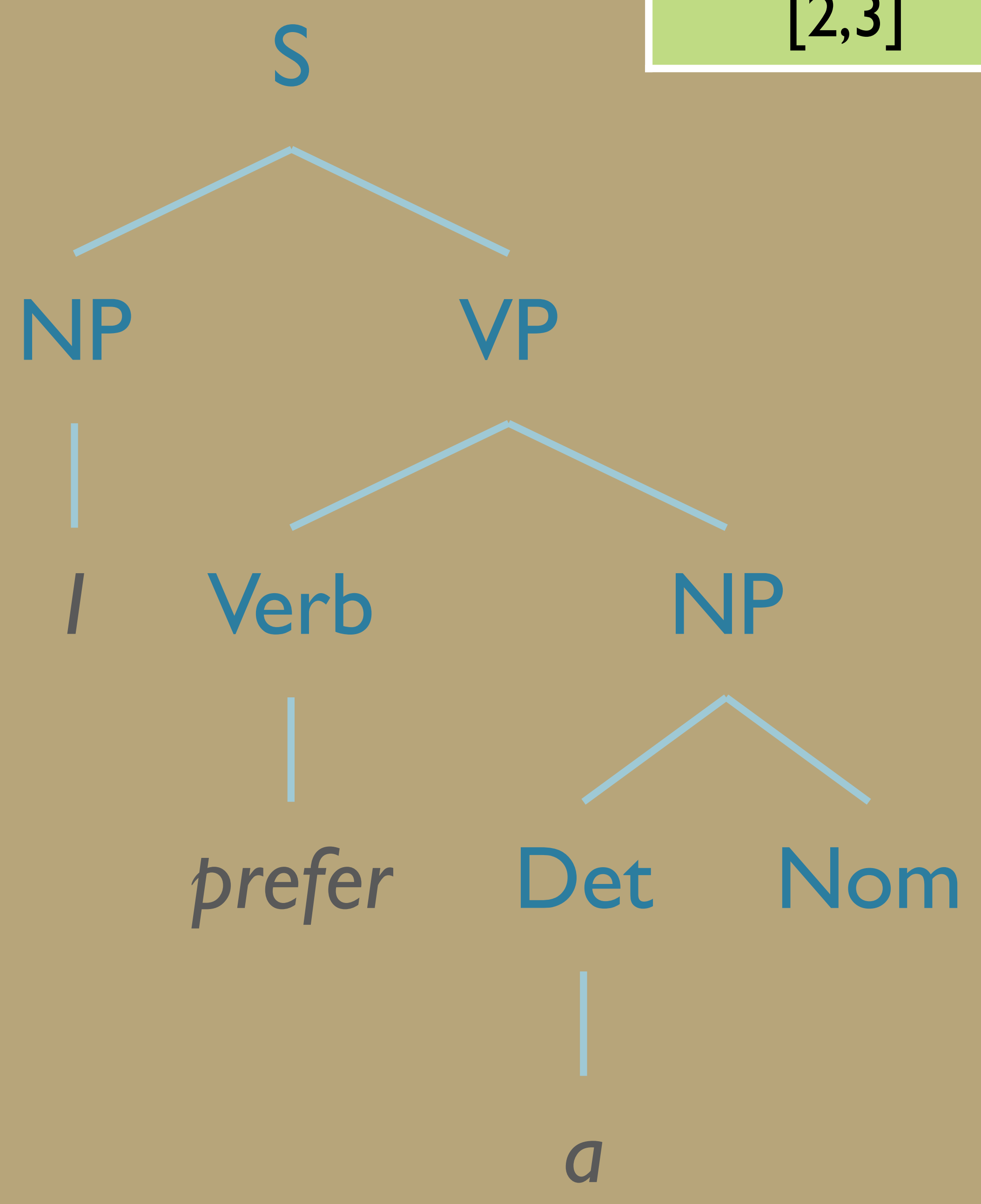
NP, Pronoun [0,1]	**S** [0,2]	[0,3]	**S** [0,4]	[0,5]	**S** [0,6]
	Verb, VP, S [1,2]	[1,3]	**VP, X2, S** [1,4]	[1,5]	**VP, X2, S** [1,6]
		Det [2,3]	**NP** [2,4]	[2,5]	**NP** [2,6]
			Noun, Nom [3,4]	[3,5]	**Nom** [3,6]
				Prep [4,5]	**PP** [4,6]
					NNP, NP [5,6]



I *prefer* *a* *flight* *on* *TWA*

```
  cky_table[0,6][S] = {(NP, (0,1),
                        VP, (1,6))}
    cky_table[0,1][NP] = {('I')}
cky_table[1,6][VP] = {(Verb, (1,2),
                       NP, (2,6)),
                      (X2, (1,4),
                       PP, (4,6))}
cky_table[1,2][Verb] = {('prefer')}
 cky_table[2,6][NP] = {(Det, (2,3),
                        Nom, (3,6)}
   cky_table[2,3][Det] = {('a')}
```

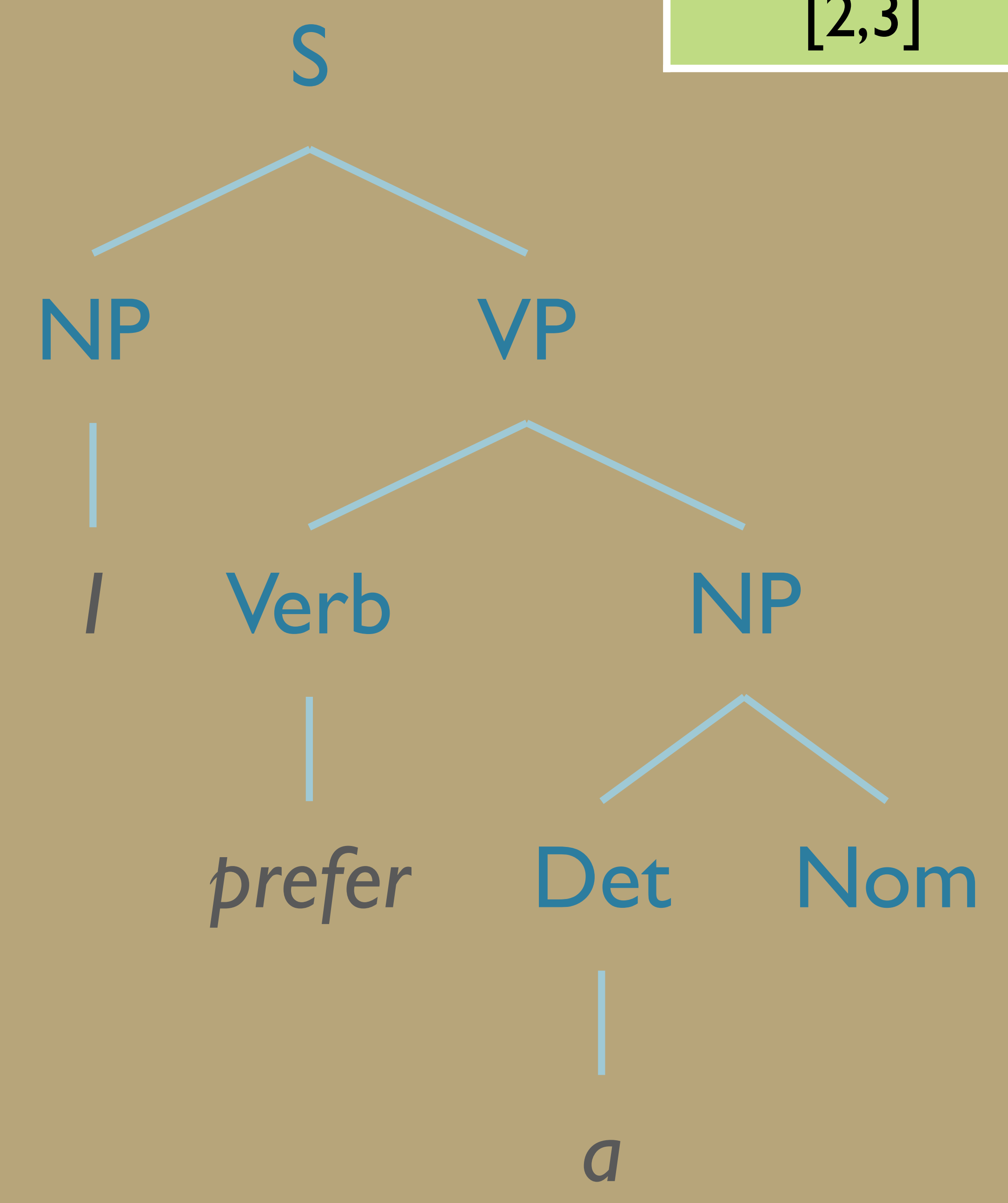
NP, Pronoun [0,1]	**S** [0,2]	[0,3]	**S** [0,4]	[0,5]	**S** [0,6]
	Verb, VP, S [1,2]	[1,3]	**VP, X2, S** [1,4]	[1,5]	**VP, X2, S** [1,6]
		Det [2,3]	**NP** [2,4]	[2,5]	**NP** [2,6]
			Noun, Nom [3,4]	[3,5]	**Nom** [3,6]
				Prep [4,5]	**PP** [4,6]
					NNP, NP [5,6]



I prefer a flight on TWA

```
  cky_table[0,6][S] = {(NP, (0,1),
                        VP, (1,6))}
    cky_table[0,1][NP] = {('I')}
cky_table[1,6][VP] = {(Verb, (1,2),
                       NP, (2,6)),
                      (X2, (1,4),
                       PP, (4,6))}
cky_table[1,2][Verb] = {('prefer')}
 cky_table[2,6][NP] = {(Det, (2,3),
                       Nom, (3,6)}
   cky_table[2,3][Det] = {('a')}
```

NP, Pronoun [0,1]	**S** [0,2]	[0,3]	**S** [0,4]	[0,5]	**S** [0,6]
	Verb, VP, S [1,2]	[1,3]	**VP, X2, S** [1,4]	[1,5]	**VP, X2, S** [1,6]
		Det [2,3]	**NP** [2,4]	[2,5]	**NP** [2,6]
			Noun, Nom [3,4]	[3,5]	**Nom** [3,6]
				Prep [4,5]	**PP** [4,6]
					NNP, NP [5,6]

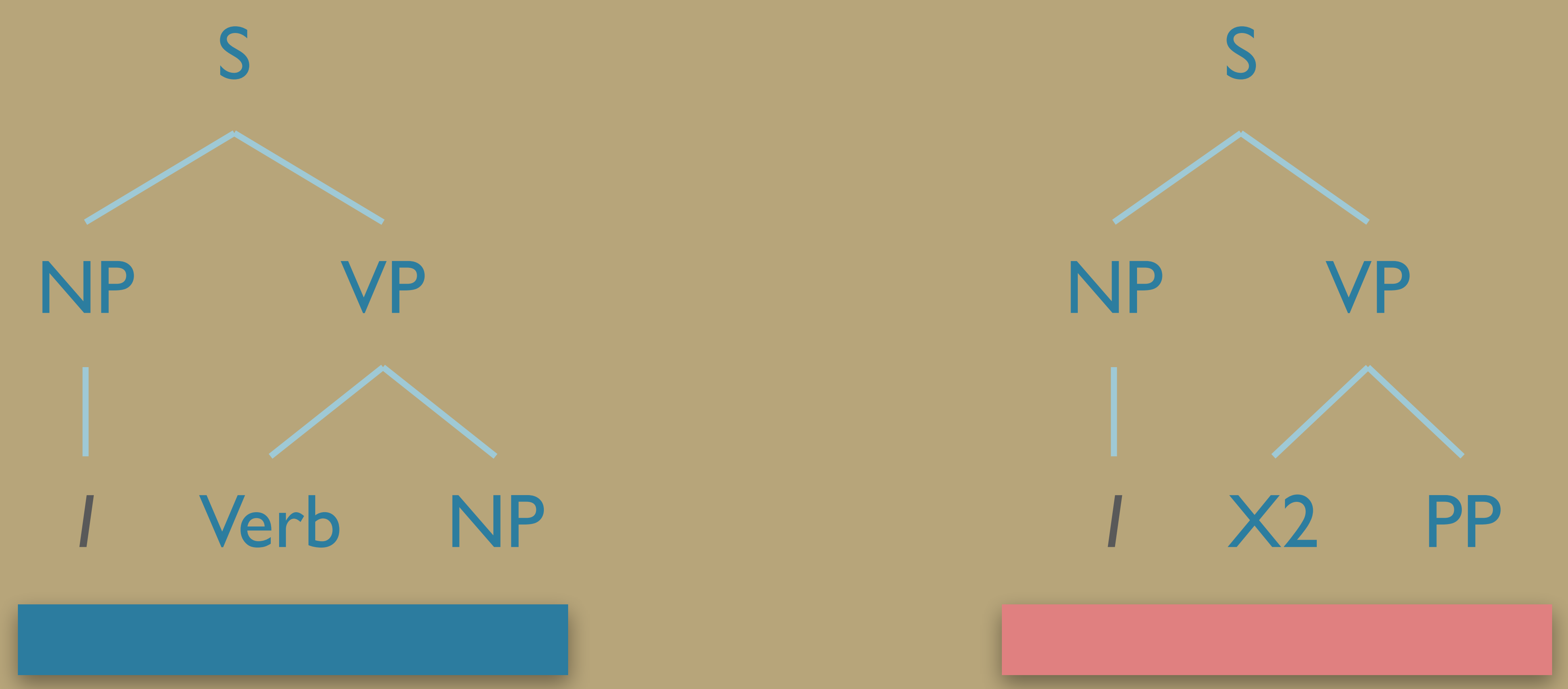


Exercise: finish the parse!

I *prefer* *a* *flight* *on* *TWA*

```
cky_table[0,6][S] = {(NP, (0,1),
                      VP, (1,6))}
   cky_table[0,1][NP] = {('I')}
cky_table[1,6][VP] = {(Verb, (1,2),
                       NP, (2,6)),
                      (X2, (1,4),
                       PP, (4,6))}
```

NP, Pronoun [0,1]	**S** [0,2]	[0,3]	**S** [0,4]	[0,5]	**S** [0,6]
	Verb, VP, S [1,2]	[1,3]	**VP, X2, S** [1,4]	[1,5]	**VP, X2, S** [1,6]
		Det [2,3]	**NP** [2,4]	[2,5]	**NP** [2,6]
			Noun, Nom [3,4]	[3,5]	**Nom** [3,6]
				Prep [4,5]	**PP** [4,6]
					NNP, NP [5,6]



I *prefer* *a* *flight* *on* *TWA*

PCFGs: Recap

PCFGs: Formal Definition

N	a set of **non-terminal symbols** (or **variables**)
Σ	a set of **terminal symbols** (disjoint from N)
R	a set of rules of productions, each of the form $A \rightarrow \beta[p]$, where A is a non-terminal where A is a non-terminal, β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$ and p is a number between 0 and 1 expressing $P(\beta \| A)$
S	a designated **start symbol**

Disambiguation

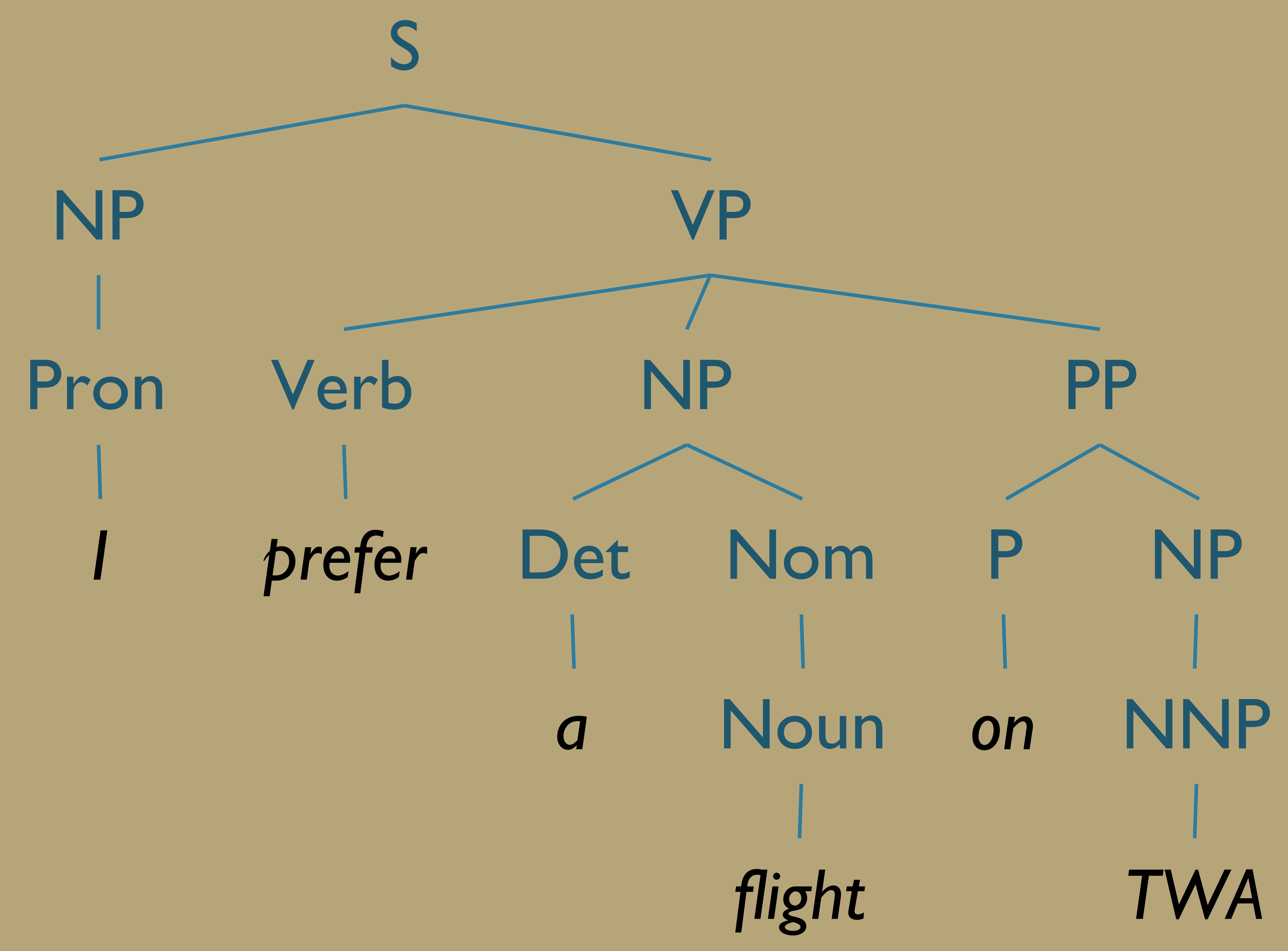
- A PCFG assigns probability to each parse tree T for input S
- Probability of T: product of all rules used to derive T

$$P(T,S) = \prod_{i=1}^{n} P(RHS_i \mid LHS_i)$$

$$P(T,S) = P(T) \cdot P(S \mid T) = P(T)$$

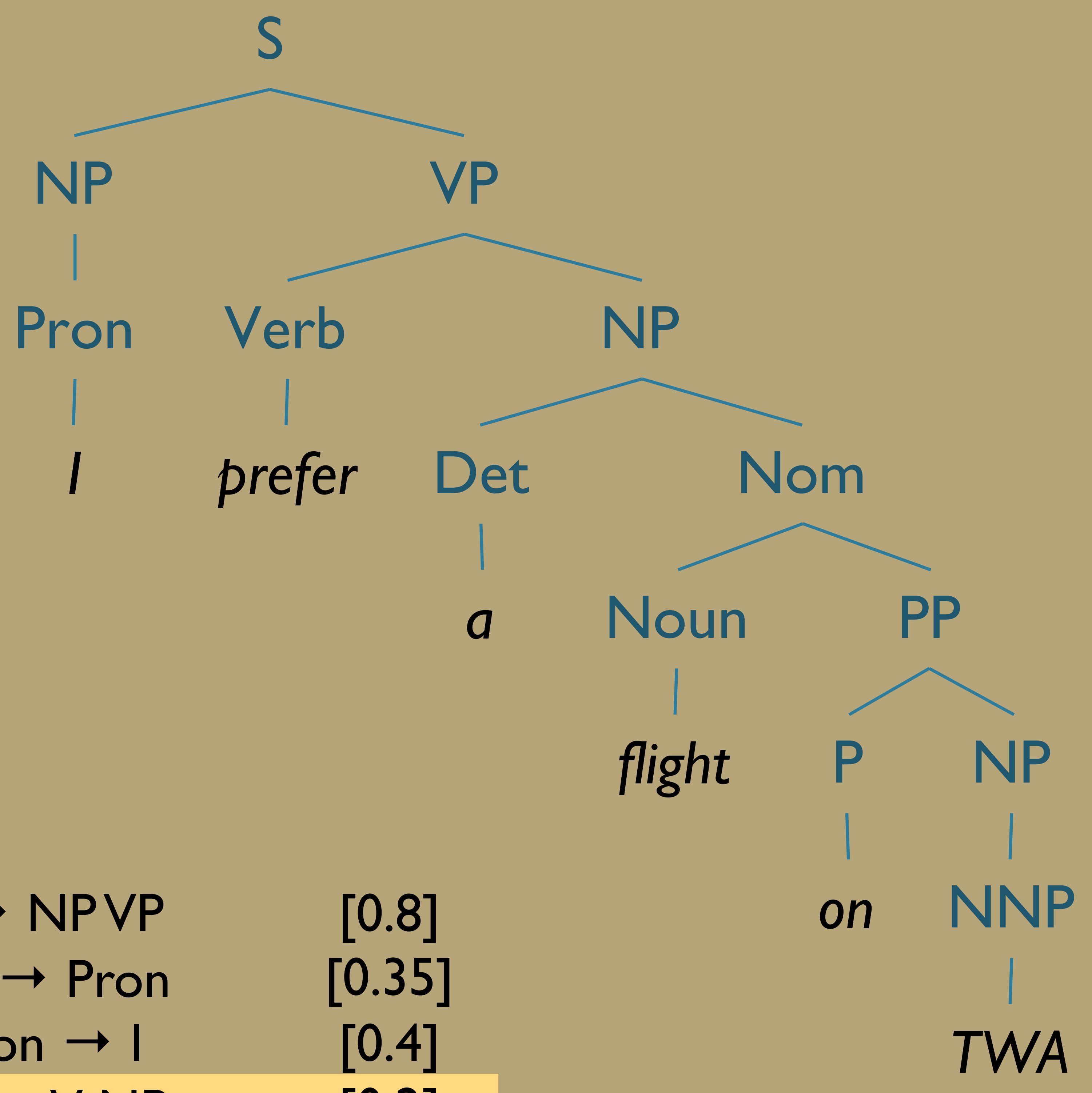
Application: Language Modeling

- *n*-grams helpful for modeling the probability of a string
- To model a whole sentence with *n-grams* either:
  - Must use 10+-grams… too sparse
  - Approximate using conditioning on limited context: $P(w_i \mid w_{i-1}) = \dfrac{P(w_{i-1}, w_i)}{P(w_{i-1})}$
- PCFGs are able to give probability of entire string without as bad sparsity
- Model probability of *syntactically valid* sentences
  - Not just probability of sequence of words



S → NP VP	[0.8]
NP → Pron	[0.35]
Pron → I	[0.4]
VP → V NP PP	[0.1]
V → prefer	[0.4]
NP → Det Nom	[0.2]
Det → a	[0.3]
Nom → N	[0.75]
N → flight	[0.3]
PP → P NP	[1.0]
P → on	[0.2]
NP → NNP	[0.3]
NNP → NWA	[0.4]

$\sim 1.452 \times 10^{-6}$



S → NP VP	[0.8]
NP → Pron	[0.35]
Pron → I	[0.4]
VP → V NP	[0.2]
V → prefer	[0.4]
NP → Det Nom	[0.2]
Det → a	[0.3]
Nom → Nom PP	[0.05]
Nom → N	[0.75]
N → flight	[0.3]
PP → P NP	[1.0]
P → on	[0.2]
NP → NNP	[0.3]
NNP → NWA	[0.4]

$\sim 1.452 \times 10^{-7}$

Parsing Problem for PCFGs

- Select T such that *(s.t.)*

$$\hat{T}(S) = \underset{T s.t. S = yield(T)}{\mathbf{argmax}} P(T)$$

- String of words S is ***yield*** of parse tree
- Select the tree $\hat{T}$ that maximizes the probability of the parse
- Extend existing algorithms: e.g. CKY

PCFGs: Parsing

Probabilistic CKY (PCKY)

- Like regular CKY
  - Assumes grammar in Chomsky Normal Form (CNF)
    - $A \rightarrow B\ C$
    - $A \rightarrow \mathrm{w}$
  - Represent input with indices b/t words:
    - $_0$ Book $_1$ that $_2$ flight $_3$ through $_4$ Houston $_5$

Probabilistic CKY (PCKY)

- For input string length n and non-terminals V
  - Cell [i, j, A] in (n+1) × (n+1) × V matrix
  - Contains probability that A spans [i, j]

PCKY Algorithm

function PROBABILISTIC-CKY-PARSE(*words, grammar*) **returns** most probable parse and its probability

for j ← **from** 1 to LENGTH(*words*) **do**

 for all { A | A → *words[j]* ∈ *grammar* }

  table[j–1, j, A] ← P(A → *words[j]*)

 for i ← **from** j–2 **downto** 0 **do**

  for k ← i + 1 **to** j–1 **do**

  for all { A | A → B C ∈ *grammar*,

   and *table*[i, k, B] > 0 **and** *table*[k, j, C] > 0 }

  if (*table*[i, j, A] < P(A → BC)×*table*[i, k, B]×*table*[k,j,C]) **then**

   table[i, j, A] ← P(A → BC)×*table*[i,k,B]×*table*[k,j,C]

   back[i, j, A] ← { k, B, C }

  return BUILD_TREE(*back*[1, LENGTH(*words*), S]), *table*[1,LENGTH(*words*), S]

PCKY Grammar Segment

S → NP VP	[0.80]		Det → the	[0.40]
NP → Det N	[0.30]		Det → a	[0.40]
VP → V NP	[0.20]		V → includes	[0.05]
			N → meal	[0.01]
			N → flight	[0.02]

PCKY Matrix

$S \rightarrow NP\ VP \quad [0.80]$

$NP \rightarrow Det\ N \quad [0.30]$

$VP \rightarrow V\ NP \quad [0.20]$

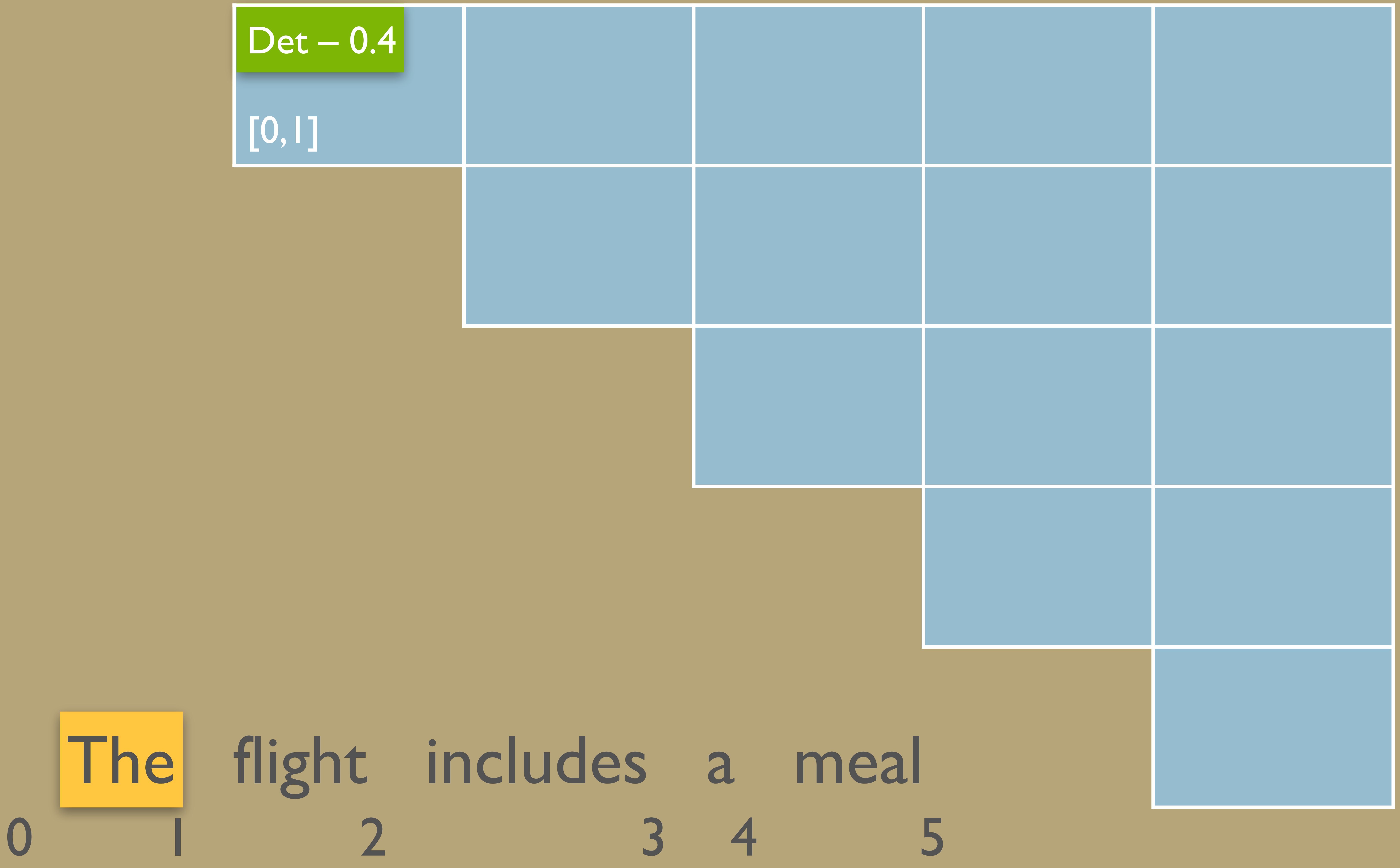
$Det \rightarrow \text{the} \quad [0.40]$

$Det \rightarrow \text{a} \quad [0.40]$

$V \rightarrow \text{includes} \quad [0.05]$

$N \rightarrow \text{meal} \quad [0.01]$

$N \rightarrow \text{flight} \quad [0.02]$



PCKY Matrix

$S \rightarrow NP\ VP \quad [0.80]$

$NP \rightarrow Det\ N \quad [0.30]$

$VP \rightarrow V\ NP \quad [0.20]$

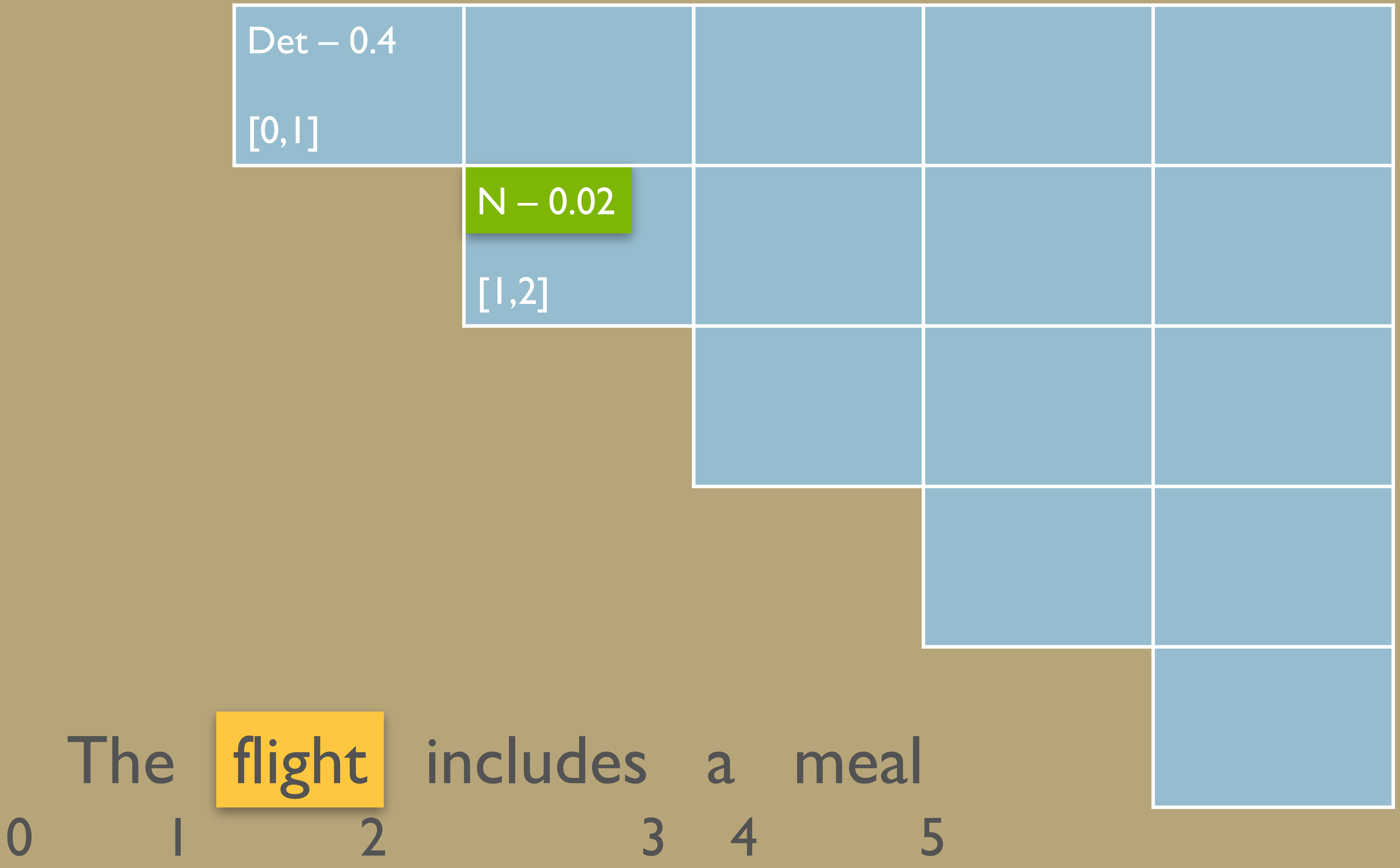
$Det \rightarrow \text{the} \quad [0.40]$

$Det \rightarrow \text{a} \quad [0.40]$

$V \rightarrow \text{includes} \quad [0.05]$

$N \rightarrow \text{meal} \quad [0.01]$

$N \rightarrow \text{flight} \quad [0.02]$



PCKY Matrix

$S \rightarrow NP\ VP \quad [0.80]$

$NP \rightarrow Det\ N \quad [0.30]$

$VP \rightarrow V\ NP \quad [0.20]$

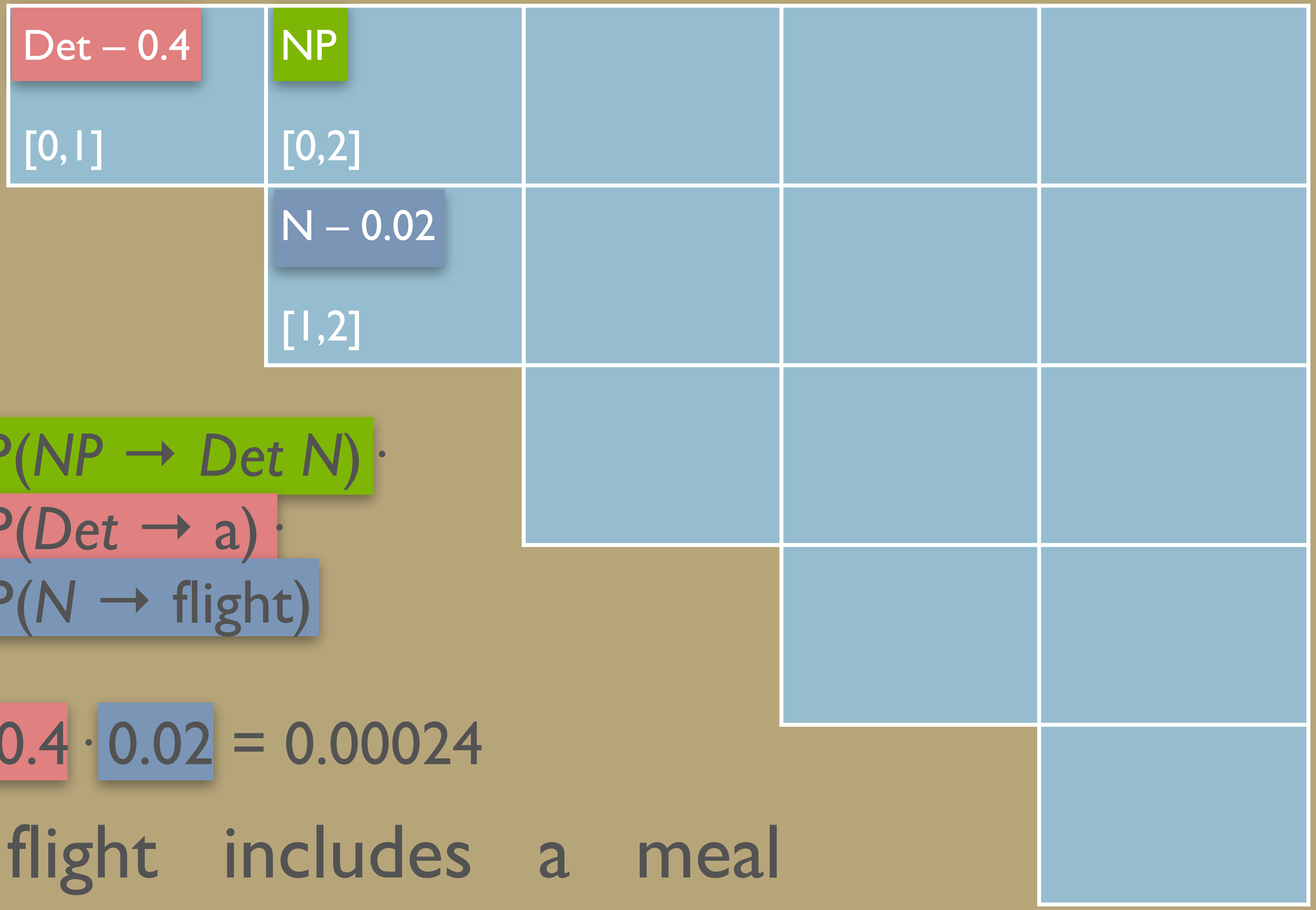
$Det \rightarrow \text{the} \quad [0.40]$

$Det \rightarrow \text{a} \quad [0.40]$

$V \rightarrow \text{includes} \quad [0.05]$

$N \rightarrow \text{meal} \quad [0.01]$

$N \rightarrow \text{flight} \quad [0.02]$



$$P = P(NP \rightarrow Det\ N) \cdot P(Det \rightarrow a) \cdot P(N \rightarrow flight)$$

$$P = 0.3 \cdot 0.4 \cdot 0.02 = 0.00024$$

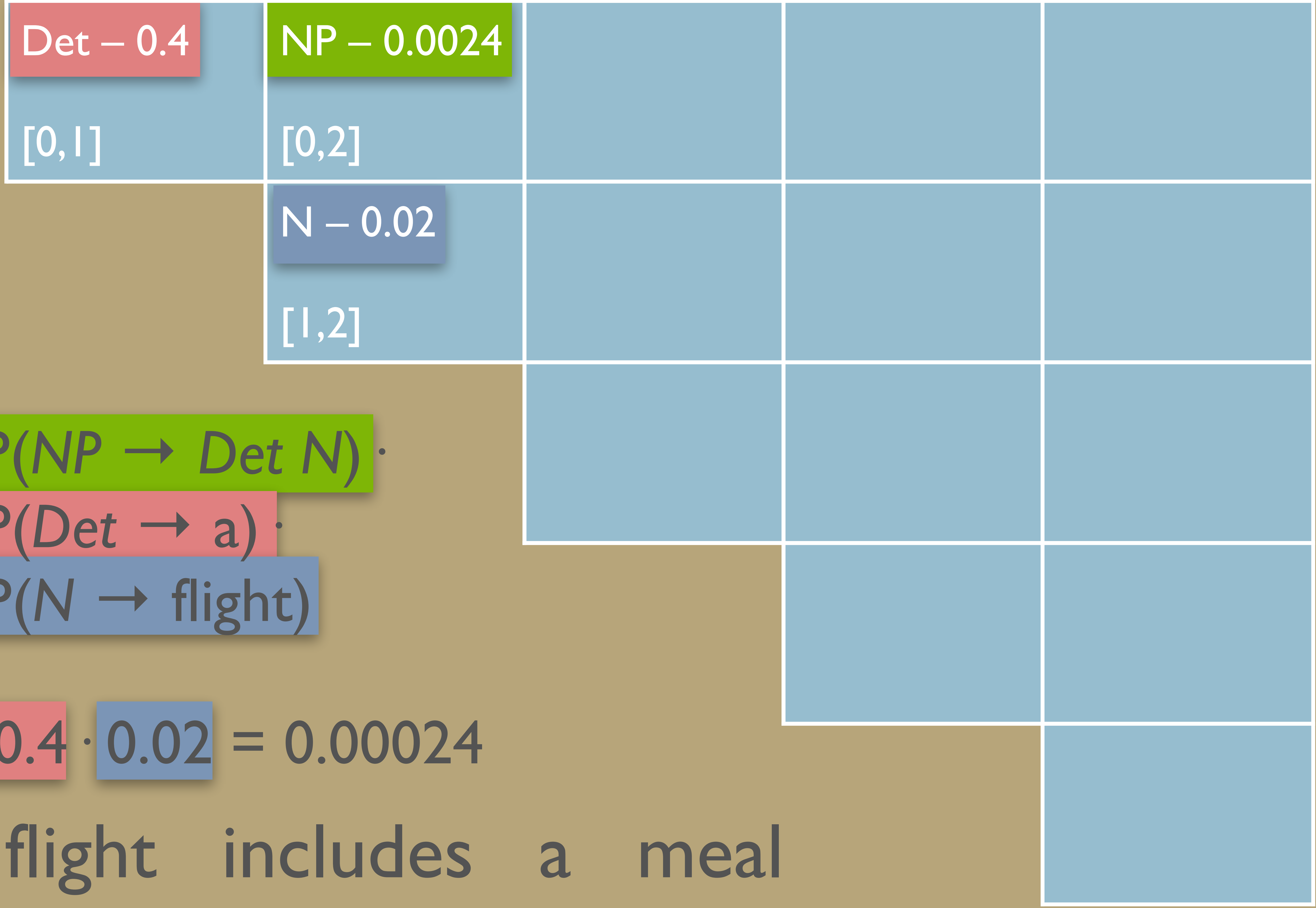
The flight includes a meal

0 1 2 3 4 5

PCKY Matrix

$S \rightarrow NP\ VP$ [0.80]
$NP \rightarrow Det\ N$ [0.30]
$VP \rightarrow V\ NP$ [0.20]

$Det \rightarrow$ the [0.40]
$Det \rightarrow$ a [0.40]
$V \rightarrow$ includes [0.05]
$N \rightarrow$ meal [0.01]
$N \rightarrow$ flight [0.02]



$$P = P(NP \rightarrow Det\ N) \cdot P(Det \rightarrow \text{a}) \cdot P(N \rightarrow \text{flight})$$

$$P = 0.3 \cdot 0.4 \cdot 0.02 = 0.00024$$

The flight includes a meal
0 1 2 3 4 5

PCKY Matrix

$S \rightarrow NP\ VP$ [0.80]
$NP \rightarrow Det\ N$ [0.30]
$VP \rightarrow V\ NP$ [0.20]

$Det \rightarrow$ the [0.40]
$Det \rightarrow$ a [0.40]
$V \rightarrow$ includes [0.05]
$N \rightarrow$ meal [0.01]
$N \rightarrow$ flight [0.02]

Det – 0.4 [0,1]	NP – 0.0024 [0,2]	[0,3]	[0,4]	S – 2.304×10^{-8} [0,5]
	N – 0.02 [1,2]	[1,3]	[1,4]	[1,5]
		V – 0.05 [2,3]	[2,4]	VP – 1.2×10^{-5} [2,5]
			Det – 0.4 [3,4]	NP – 0.0012 [3,5]
				N – 0.01 [4,5]

The flight includes a meal

0 1 2 3 4 5

Inducing a PCFG

Learning Probabilities

- Simplest way:
  - Use treebank of parsed sentences
  - To compute probability of a rule, count:
    - Number of times a nonterminal is expanded: $\Sigma_\gamma \; Count(\alpha \rightarrow \gamma)$
    - Number of times a nonterminal is expanded by a given rule: $Count(\alpha \rightarrow \beta)$

$$P(\alpha \rightarrow \beta \,|\, \alpha) = \frac{Count(\alpha \rightarrow \beta)}{\sum_\gamma Count(\alpha \rightarrow \gamma)} = \frac{Count(\alpha \rightarrow \beta)}{Count(\alpha)}$$

- Alternative: Learn probabilities by re-estimating
  - (Later)

Probabilistic Parser Development Paradigm

	Train	Dev	Test
Size	Large (eg. WSJ 2–21, 39,830 sentences)	Small (e.g. WSJ 22)	Small/Med (e.g. WSJ, 23, 2,416 sentences)
Usage	Estimate rule probabilities	Tuning/Verification, Check for Overfit	Held Out, Final Evaluation

Parser Evaluation

Parser Evaluation

- Assume a ‘gold standard’ set of parses for test set
- How can we tell how good the parser is?
- How can we tell how good a parse is?
  - Maximally strict: identical to ‘gold standard’
  - Partial credit:
    - Constituents in output match those in reference
      - Same start point, end point, non-terminal symbol

Parseval

- How can we compute parse score from constituents?
- Multiple Measures:

$$\text{Labeled Recall (LR)} = \frac{\text{\# of } \textbf{correct} \text{ constituents in } \textbf{hypothetical} \text{ parse}}{\text{\# of } \textbf{total} \text{ constituents in } \textbf{reference} \text{ parse}}$$

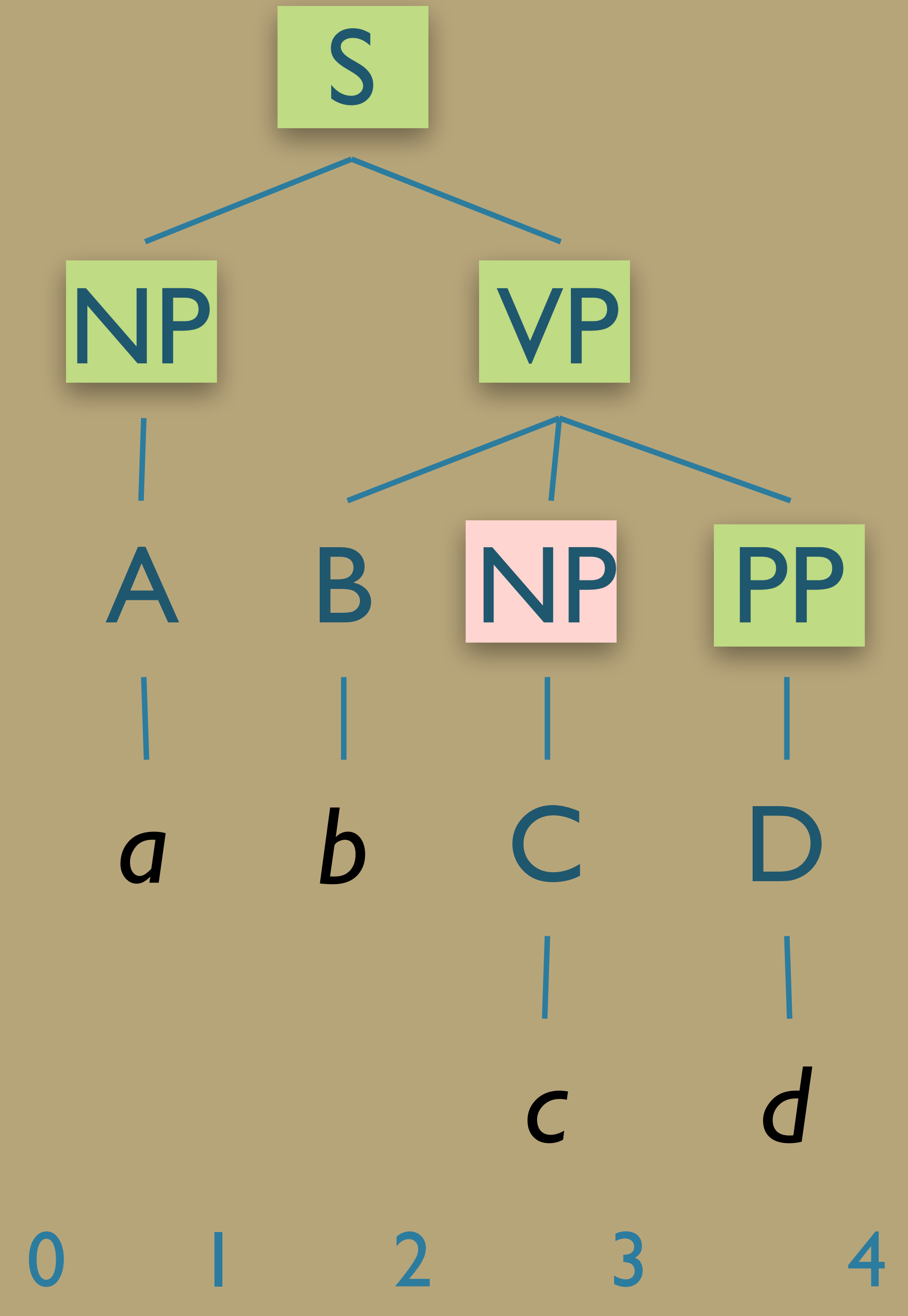
$$\text{Labeled Precision (LP)} = \frac{\text{\# of } \textbf{correct} \text{ constituents in } \textbf{hypothetical} \text{ parse}}{\text{\# of } \textbf{total} \text{ consituents in } \textbf{hypothetical} \text{ parse}}$$

Parseval

- **F-measure:**
  - Combines precision and recall
  - Let $\beta \in \mathbb{R}$, $\beta > 0$ that adjusts P vs. R s.t. $\beta \propto \frac{R}{P}$
  - F_β -measure is then: $$F_\beta = (1+\beta^2) \cdot \frac{P \cdot R}{\beta^2 \cdot P + R}$$
  - With F1-measure as $$F_1 = \frac{2PR}{P+R}$$

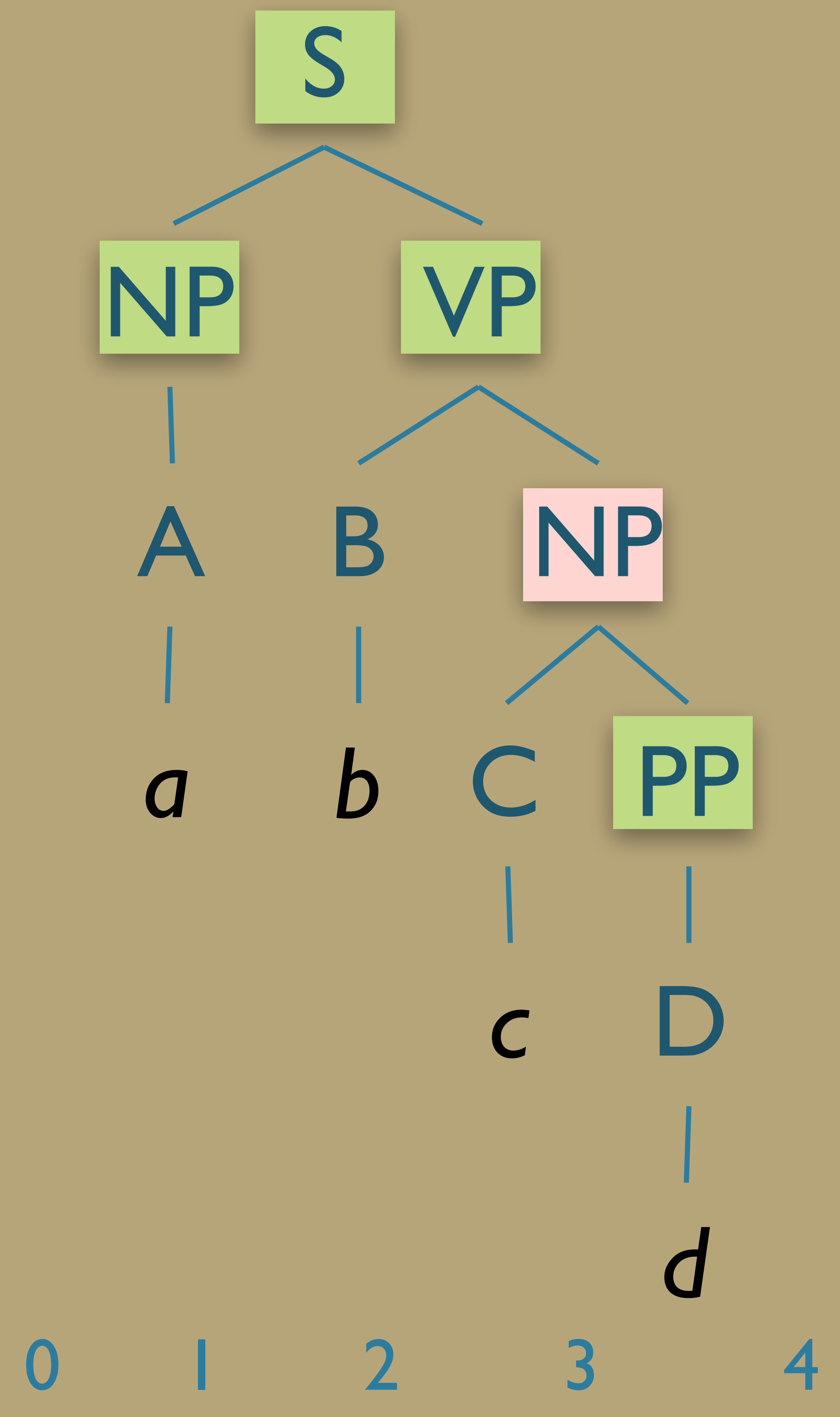
Evaluation: Example

Reference



S(0,4)
NP(0,1)
VP(1,4)
NP(2,3)
PP(3,4)

Hypothesis

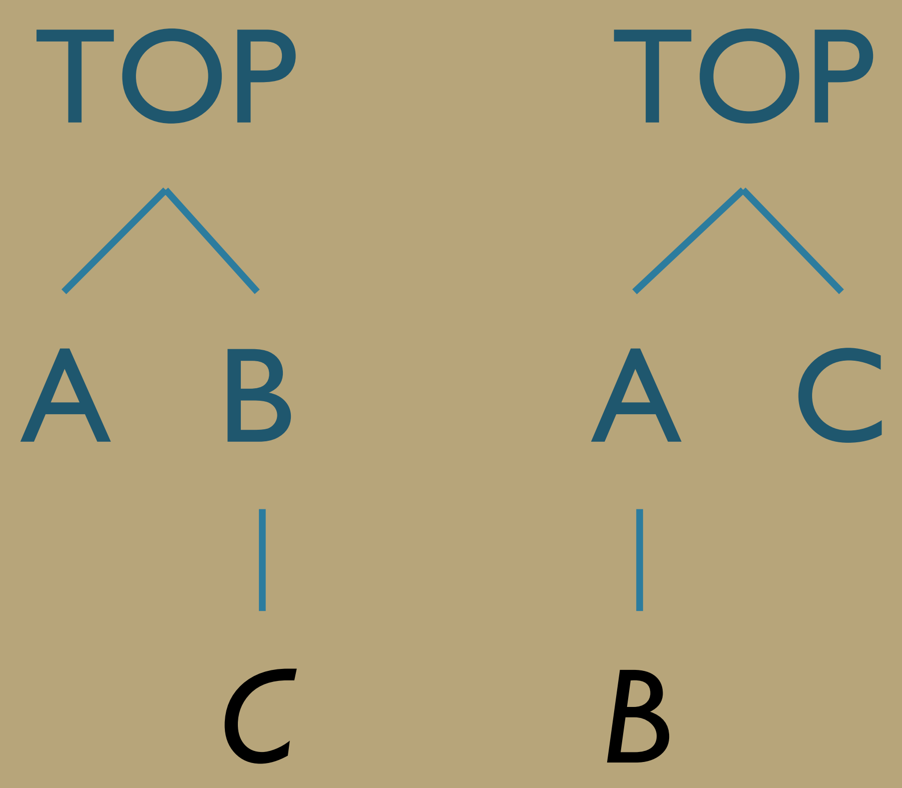


S(0,4)
NP(0,1)
VP(1,4)
NP(2,4)
PP(3,4)

LP:	4/5
LR:	4/5
F_1:	4/5

Parser Evaluation

- Crossing Brackets:
  - # of constituents where produced parse has bracketings that overlap for the siblings:
  - ((A B) C) — { (0,2), (2,3) }
    and hyp. has
    (A (B C)) — { (0,1), (1, 3) }




```c
/* crossing is counted based on the brackets */
/* in test rather than gold file (by Mike)   */
for(j=0;j<bn2;j++){
  for(i=0;i<bn1;i++){
    if(bracket1[i].result != 5 &&
       bracket2[j].result != 5 &&
       ((bracket1[i].start < bracket2[j].start &&
         bracket1[i].end   > bracket2[j].start &&
         bracket1[i].end   < bracket2[j].end) ||
        (bracket1[i].start > bracket2[j].start &&
         bracket1[i].start < bracket2[j].end &&
         bracket1[i].end   > bracket2[j].end))){
```

from evalb.c

State-of-the-Art Parsing

- Parsers trained/tested on Wall Street Journal PTB
  - LR: 90%+;
  - LP: 90%+;
  - Crossing brackets: 1%

- Standard implementation of Parseval:
  - **`evalb`**

Evaluation Issues

- Only evaluating constituency
- There are other grammar formalisms:
  - LFG (Constraint-based)
  - Dependency Structure
- **Extrinsic** evaluation
  - How well does getting the correct parse match the semantics, etc?

Earley Parsing

Earley vs. CKY

- CKY doesn't capture full original structure
  - Can back-convert binarization, terminal conversion
  - Unit non-terminals require change in CKY
- Earley algorithm
  - Supports parsing efficiently with arbitrary grammars
  - Top-down search
  - Dynamic programming
    - Tabulated partial solutions
  - Some bottom-up constraints

Earley Algorithm

- Another dynamic programming solution
  - Partial parses stored in “chart”
  - Compactly encodes ambiguity
  - $O(N^3)$
- Chart entries contain:
  - Subtree for a single grammar rule
  - Progress in completing subtree
  - Position of subtree w.r.t. input

Earley Algorithm

- First, left-to-right pass fills out a chart with N+1 states
  - Chart entires — sit between words in the input string
  - Keep track of states of the parse at those positions
  - For each word position, chart contains set of states representing all partial parse trees generate so far
    - e.g. `chart[0]` contains all partial parse trees generated at the beginning of sentence

Chart Entries

- Three types of constituents:
  - Predicted constituents
  - In-progress constituents
  - Completed constituents

Parse Progress

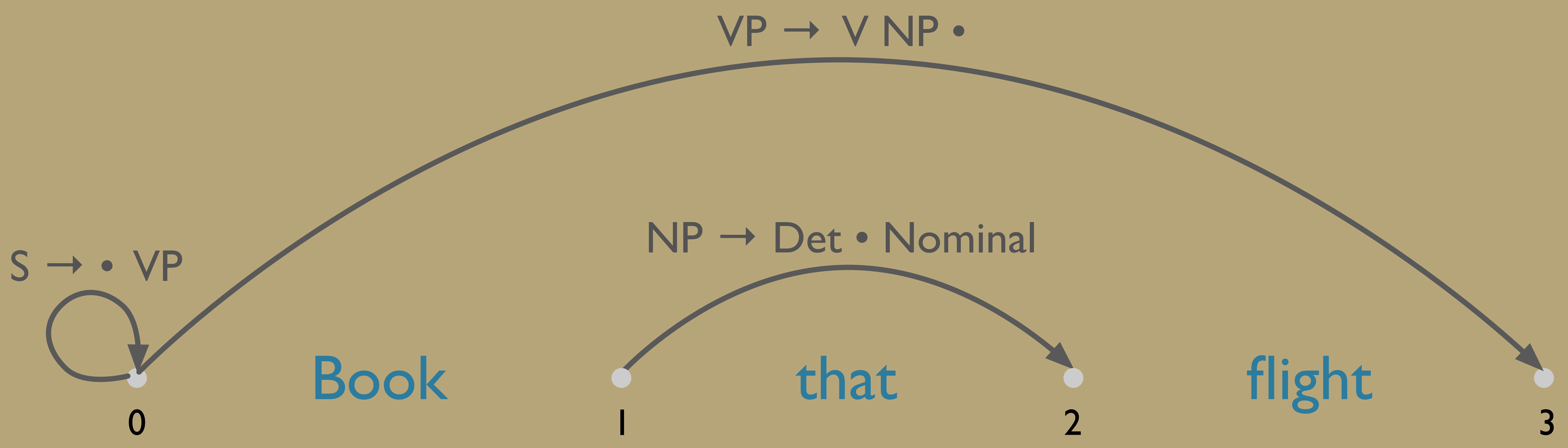
- Represented by Dotted Rules
  - Position of • indicates type of constituent
- $_0$ Book $_1$ that $_2$ flight $_3$
  - S → • VP [0,0] (predicted)
  - NP → Det • Nom [1,2] (in progress)
  - VP → V NP • [0,3] (completed)
- [x,y] tells us what portion of the input is spanned so far by rule
- Each state s_i: <dotted rule>, [<back pointer>, <current position>]

$_0$ Book $_1$ that $_2$ flight $_3$

- S → VP, [0,0]
  - First 0 means S constituent begins at the start of input
  - Second 0 means the dot is here too
  - So, this is a top-down prediction
- NP → Det • Nom, [1,2]
  - the NP begins at position 1
  - the dot is at position 2
  - so, Det has been successfully parsed
  - Nom predicted next

$_0$ Book $_1$ that $_2$ flight $_3$ (continued)

- V → V NP • [0,3]
  - Successful VP parse of entire input



Successful Parse

- Final answer found by looking at last entry in chart
- If entry resembles $S \rightarrow \alpha \bullet [0,N]$ then input parsed successfully
- Chart will also contain record of all possible parses of input string, given the grammar

Parsing Procedure for the Earley Algorithm

- Move through each set of states in order, applying one of three operations:
  - **predictor**: add predictions to the chart
  - **scanner**: read input and add corresponding state to chart
  - **completer**: move dot to right when new constituent found
- Results (new states) added to current or next set of states in chart
- No backtracking and no states removed: keep complete history of parse

Earley Algorithm

```
function EARLEY-PARSE(words, grammar) returns chart
  ENQUEUE((γ ⟶ • S, [0,0]), chart[0])
  for i ⟵ from 0 to LENGTH(words) do
    for each state in chart[i] do
      if INCOMPLETE?(state) and
          NEXT-CAT(state) is not a part of speech then
        PREDICTOR(state)
      elseif INCOMPLETE?(state) and
          NEXT-CAT(state) is a part of speech then
        SCANNER(state)
      else
        COMPLETER(state)
      end
    end
  return(chart)
```

Earley Algorithm

procedure PREDICTOR($(A \rightarrow \alpha \bullet B\ \beta\ ,\ [i,j])$)
  for each ($B \rightarrow \gamma$) **in** GRAMMAR-RULES-FOR(B,*grammar*) **do**
    ENQUEUE($(B \rightarrow \bullet\ \gamma,\ [j,j]),\ chart[j]$)
  end

procedure SCANNER($(A \rightarrow \alpha \bullet B\ \beta, [i,j])$)
  if $B \subset$ PARTS-OF-SPEECH(*word[j]*) **then**
    ENQUEUE($(B \rightarrow word[j]\ \bullet,\ [j,j+1]),\ chart[j+1]$)

procedure COMPLETER($(B \rightarrow \gamma\ \bullet,\ [j,k])$)
  for each ($A \rightarrow \alpha \bullet B\ \beta,\ [i,j]$) **in** *chart[j]* **do**
    ENQUEUE($(A \rightarrow \alpha\ B \bullet\ \beta,\ [i,k]),\ chart[k]$)
  end

3 Main Subroutines of Earley

- Predictor
  - Adds predictions into the chart
- Scanner
  - Reads the input words and enters states representing those words into the chart
- Completer
  - Moves the dot to the right when new constituents are found

Predictor

- Intuition:
  - Create new state for top-down prediction of new phrase
- Applied when non part-of-speech non-terminals are to the right of a dot:
  - S → • VP [0,0]
- Adds new states to *current* chart
  - One new state for each expansion of the non-terminal in the grammar
    VP → • V [0,0]
    VP → • V NP [0,0]

Chart[0]

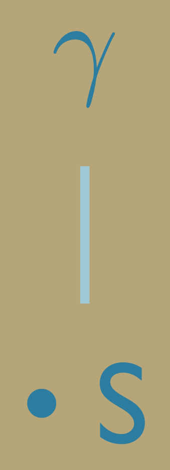
S0	$\gamma \rightarrow \cdot$ S	[0,0]	Dummy start state
S1	S $\rightarrow \cdot$ NP VP	[0.0]	Predictor
S2	S $\rightarrow \cdot$ Aux NP VP	[0,0]	Predictor
S3	S $\rightarrow \cdot$ VP	[0,0]	Predictor
S4	NP $\rightarrow \cdot$ Pronoun	[0,0]	Predictor
S5	NP $\rightarrow \cdot$ Proper-Noun	[0,0]	Predictor
S6	NP $\rightarrow \cdot$ Det Nominal	[0,0]	Predictor
S7	VP $\rightarrow \cdot$ Verb	[0,0]	Predictor
S8	VP $\rightarrow \cdot$ Verb NP	[0,0]	Predictor
S9	VP $\rightarrow \cdot$ Verb NP PP	[0,0]	Predictor
S10	VP $\rightarrow \cdot$ Verb PP	[0,0]	Predictor
S11	VP $\rightarrow \cdot$ VP PP	[0,0]	Predictor

Chart[1]

S12	Verb → book •	[0,1]	Scanner
S13	VP → Verb •	[0,1]	Completer
S14	VP → Verb • NP	[0,1]	Completer
S15	VP → Verb • NP PP	[0,1]	Completer
S16	VP → Verb • PP	[0,1]	Completer
S17	S → VP •	[0,1]	Completer
S18	VP → VP • PP	[0,1]	Completer
S19	NP → • Pronoun	[1,1]	Predictor
S20	NP → • Proper-Noun	[1,1]	Predictor
S21	NP → • Det Nominal	[1,1]	Predictor
S22	PP → • Prep NP	[1,1]	Predictor

Book that flight

S0: γ → • S [0,0]



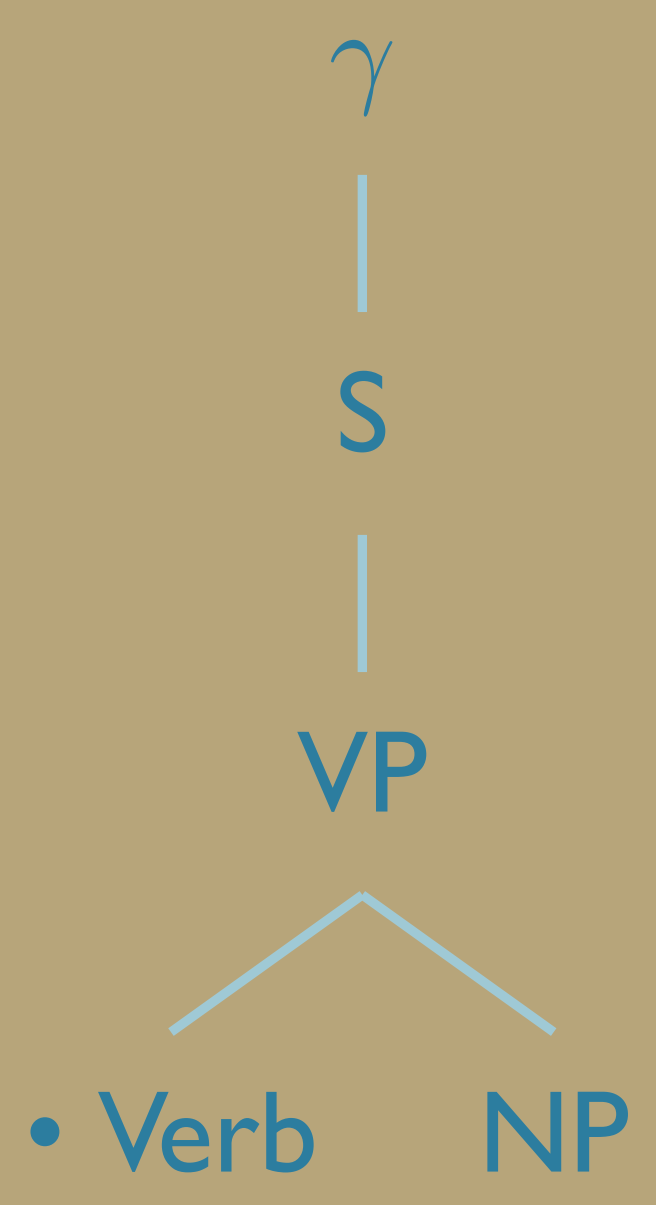
Book that flight

S0: $\gamma \rightarrow \bullet$ S [0,0]
S3: S $\rightarrow \bullet$ VP [0,0]

γ
|
S
|
$\bullet$ VP

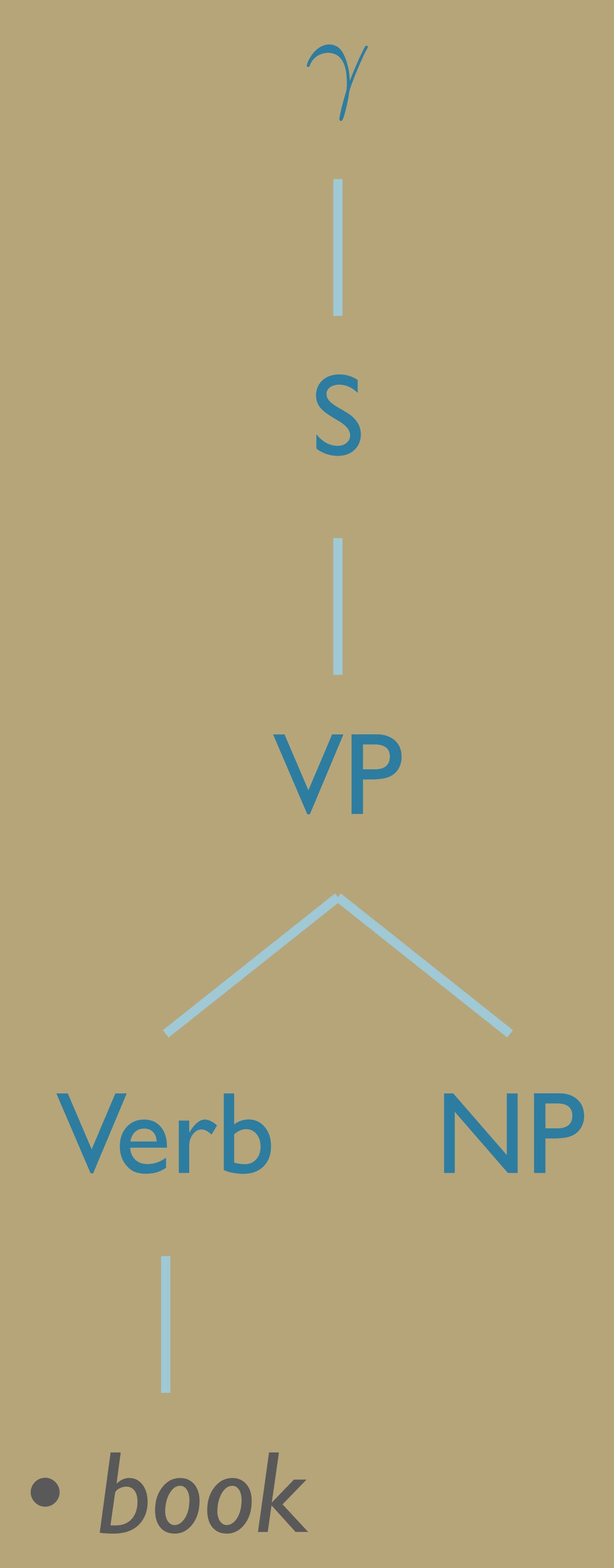
Book that flight

S0: γ → • S [0,0]
S3: S → • VP [0,0]
S8: VP → • Verb NP [0,0]



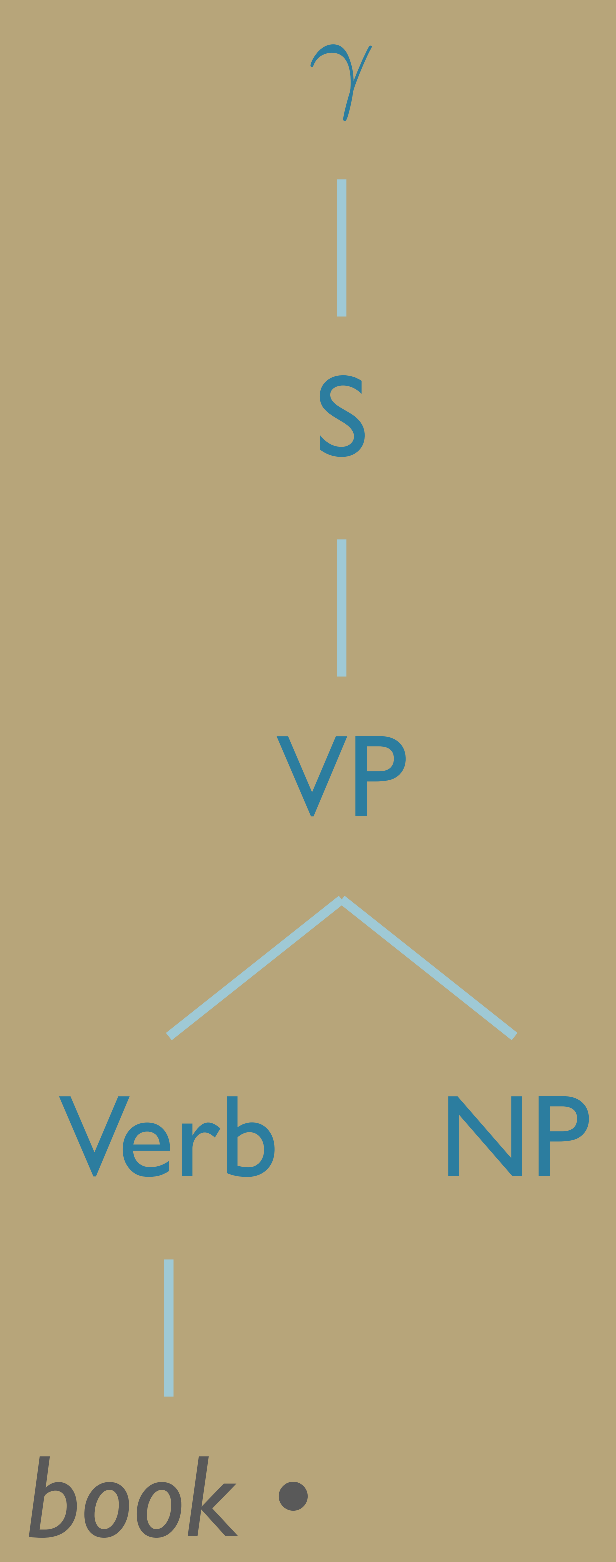
Book that flight

S0: $\gamma \rightarrow \bullet$ S [0,0]
S3: S $\rightarrow \bullet$ VP [0,0]
S8: VP $\rightarrow \bullet$ Verb NP [0,0]
S12: Verb $\rightarrow \bullet$ book [0,0]



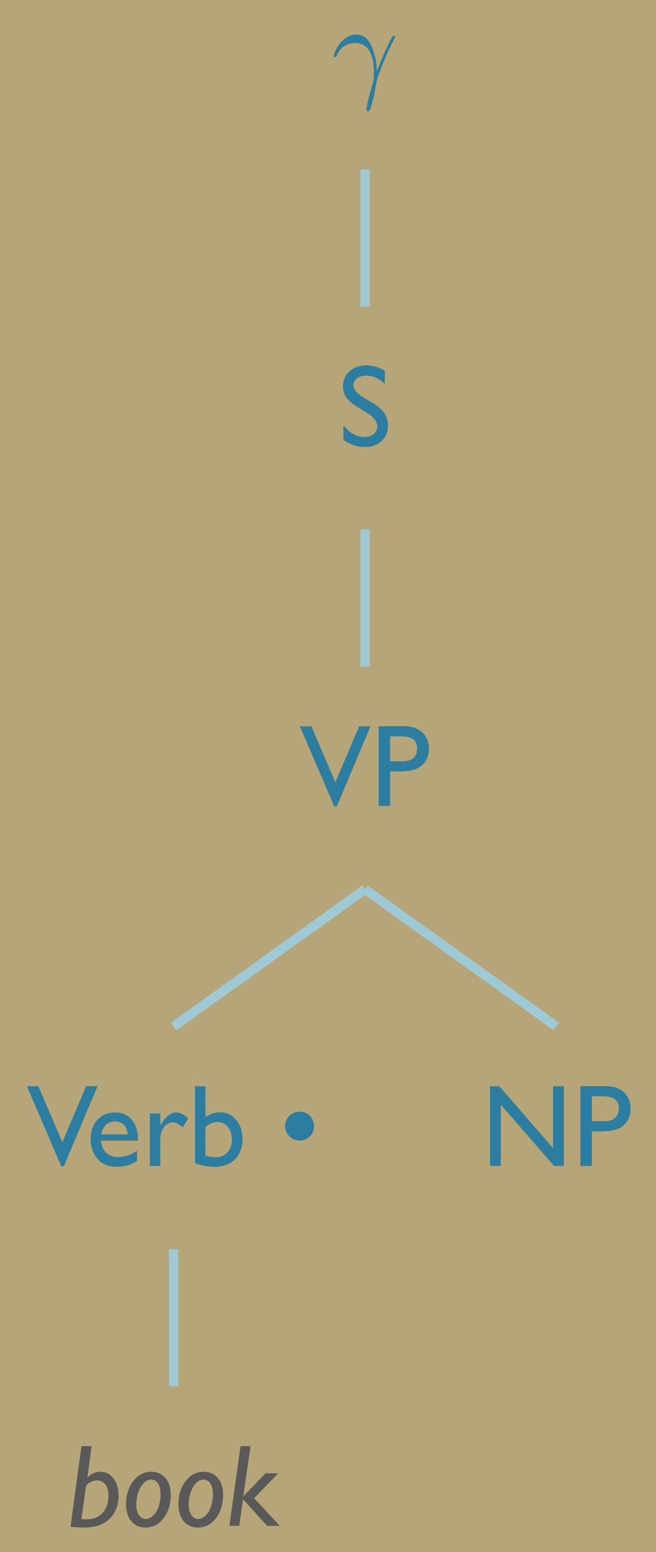
Book that flight

S0: $\gamma \rightarrow \bullet$ S [0,0]
S3: S $\rightarrow \bullet$ VP [0,0]
S8: VP $\rightarrow \bullet$ Verb NP [0,0]
S12: Verb $\rightarrow$ book $\bullet$ [0,1]



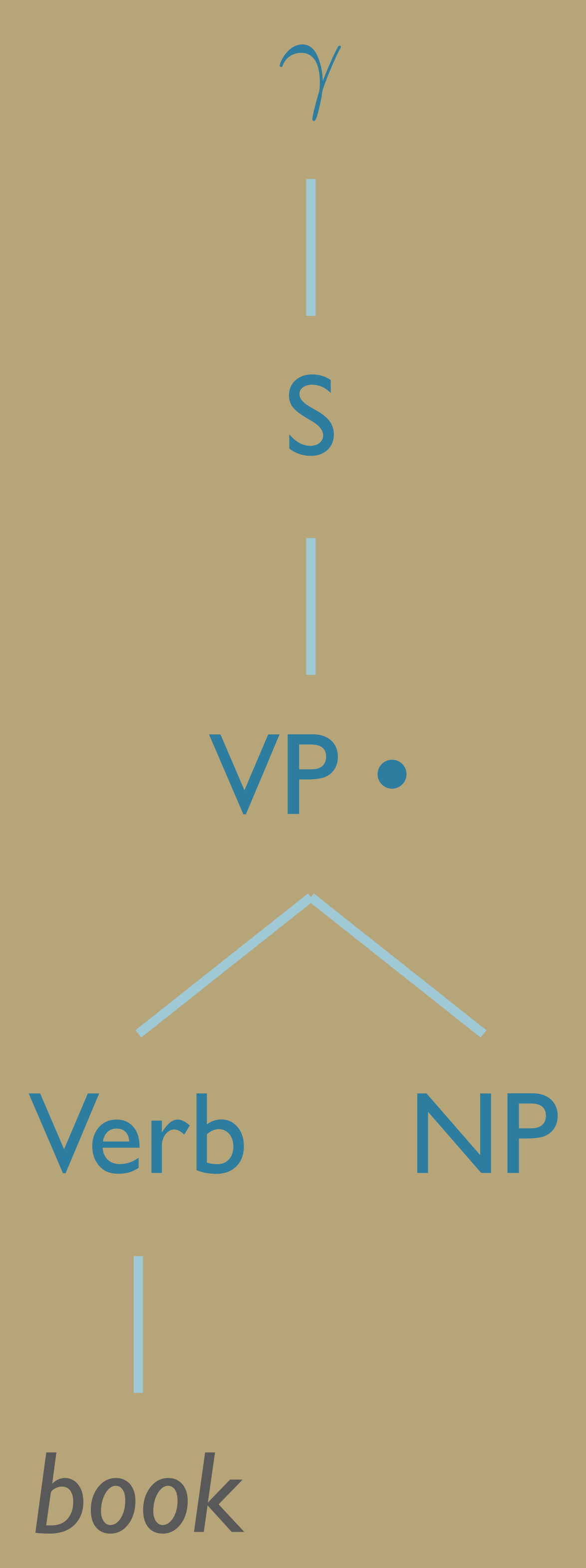
Book that flight

S0: $\gamma \rightarrow \bullet$ S [0,0]
S3: S $\rightarrow \bullet$ VP [0,0]
S8: VP $\rightarrow$ Verb $\bullet$ NP [0,1]



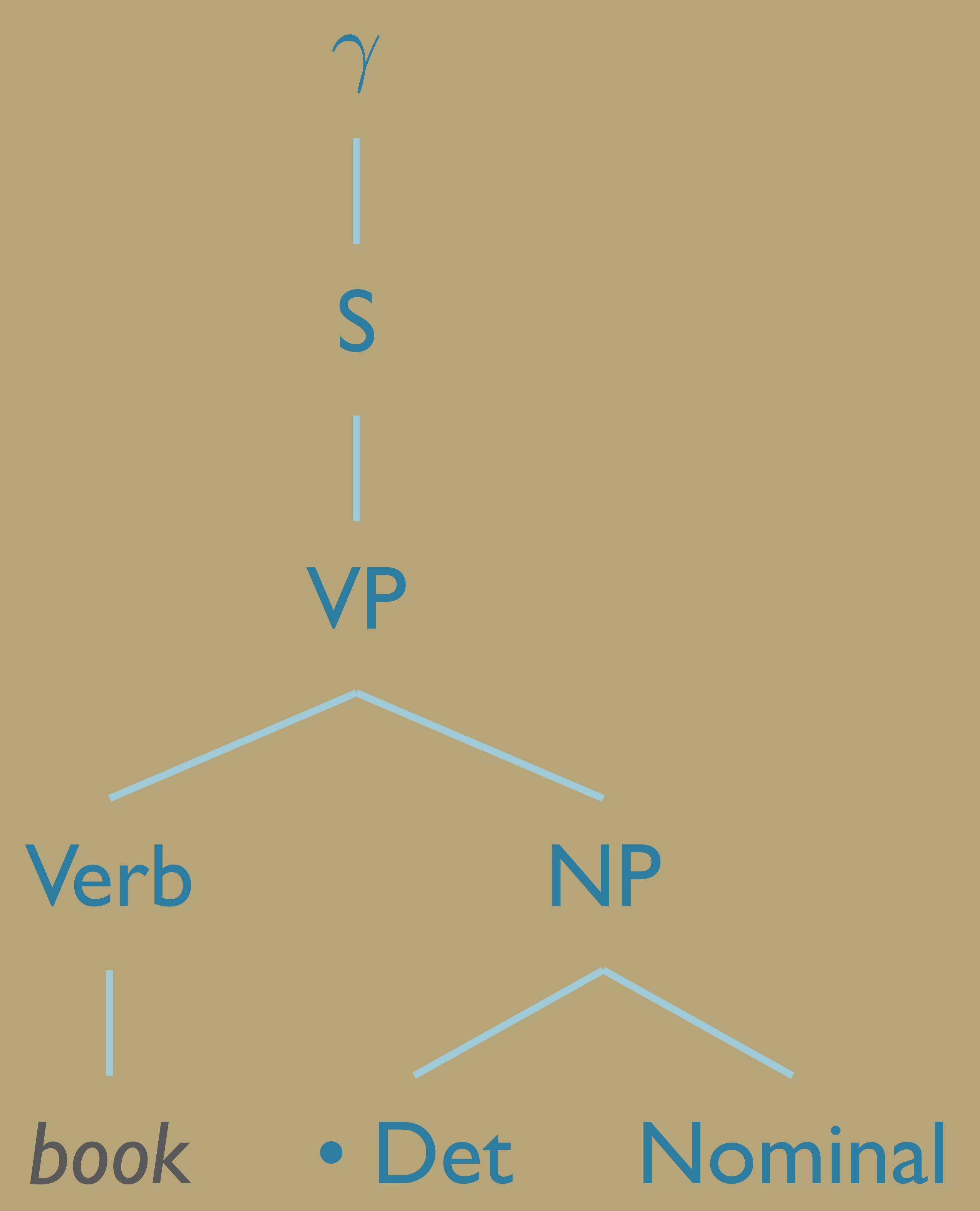
Book that flight

S0: $\gamma \rightarrow \bullet$ S [0,0]
S3: S $\rightarrow$ VP $\bullet$ [0,1]
S8: VP $\rightarrow$ Verb $\bullet$ NP [0,1]



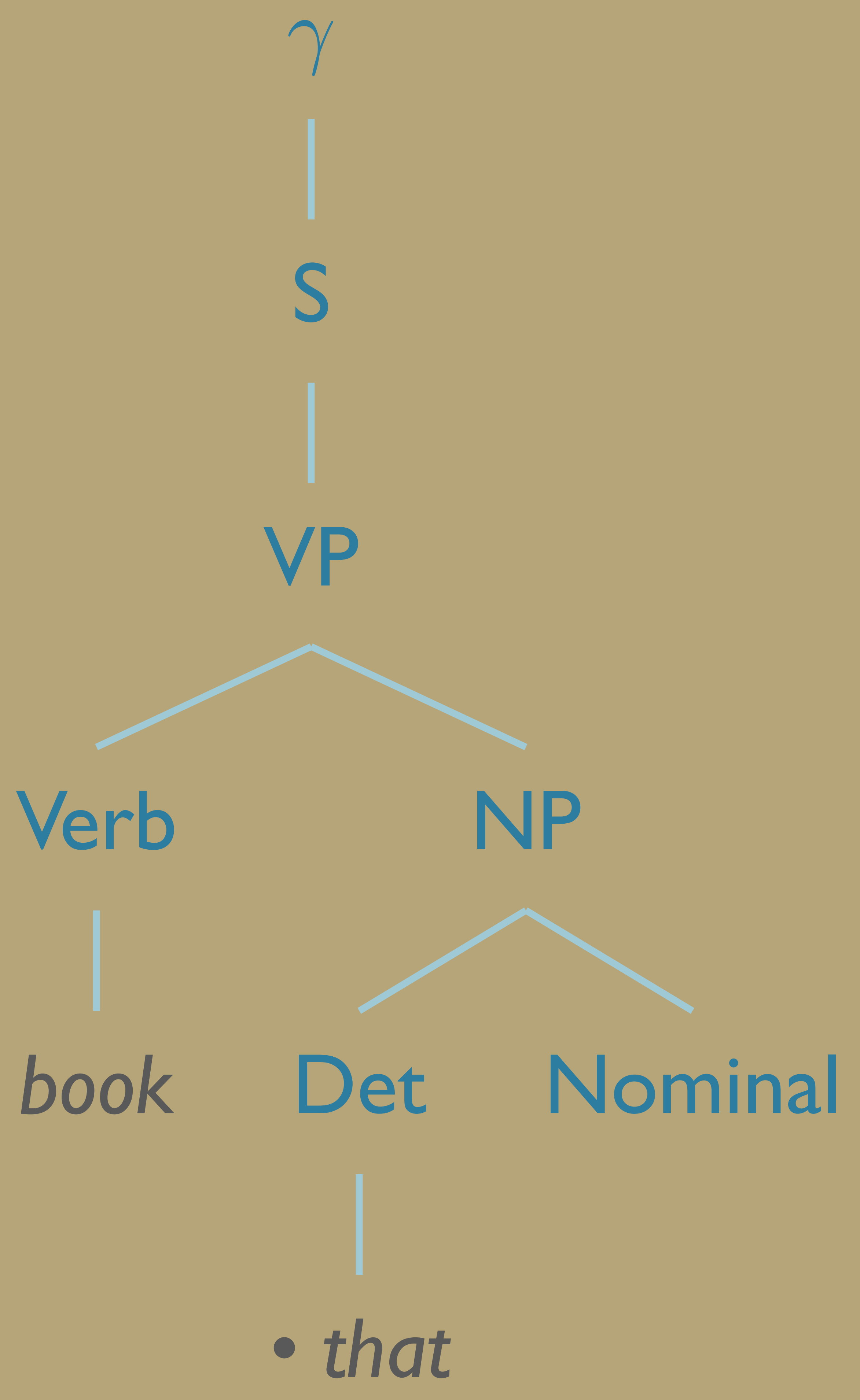
Book that flight

S0: γ → • S [0,0]
S3: S → VP • [0,1]
S8: VP → Verb • NP [0,1]
S21: NP → • Det Nominal [1,1]



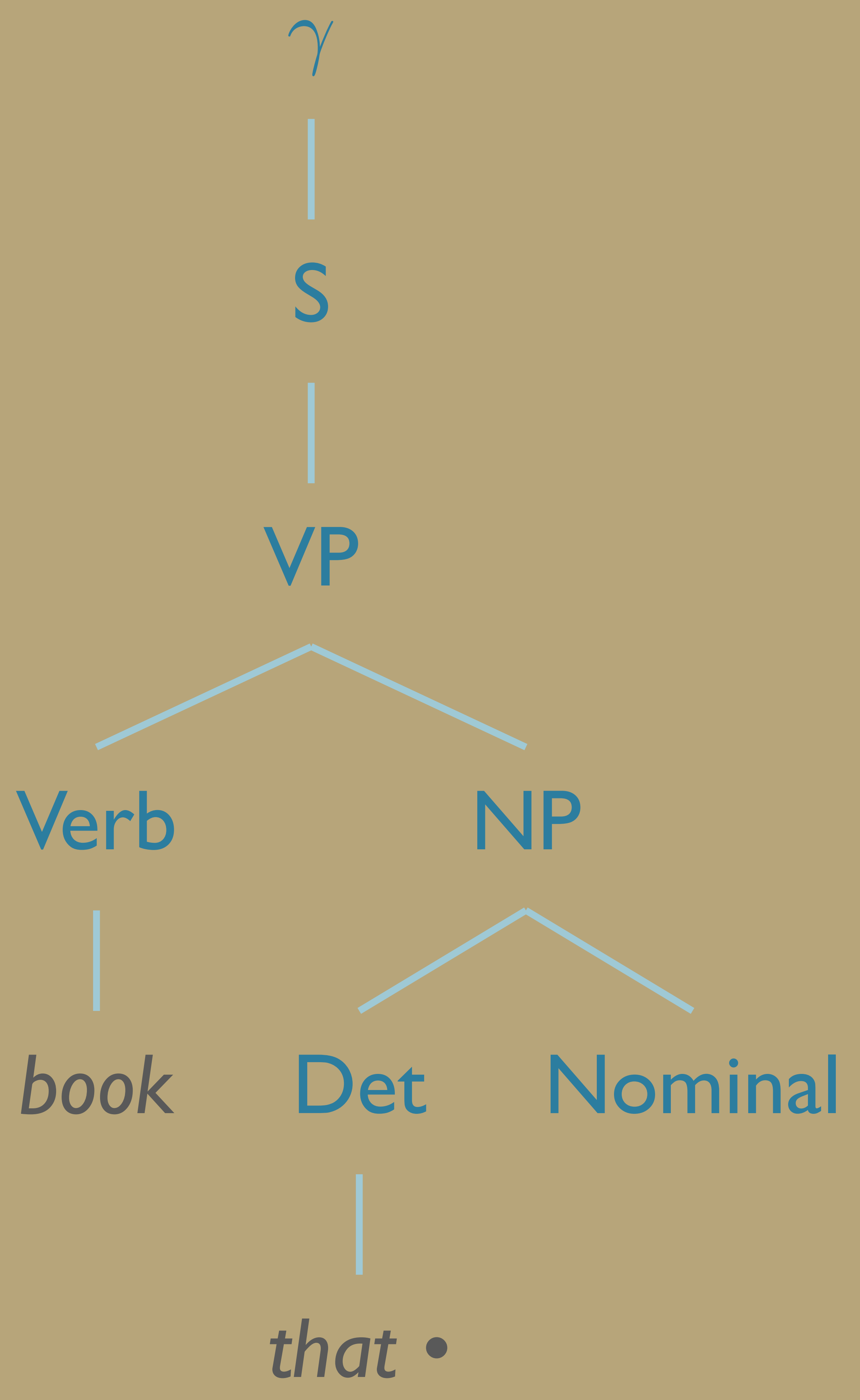
Book that flight

S0: γ → • S [0,0]
S3: S → VP • [0,1]
S8: VP → Verb • NP [0,1]
S21: NP → • Det Nominal [1,1]
S23: Det → • “that” [1,1]



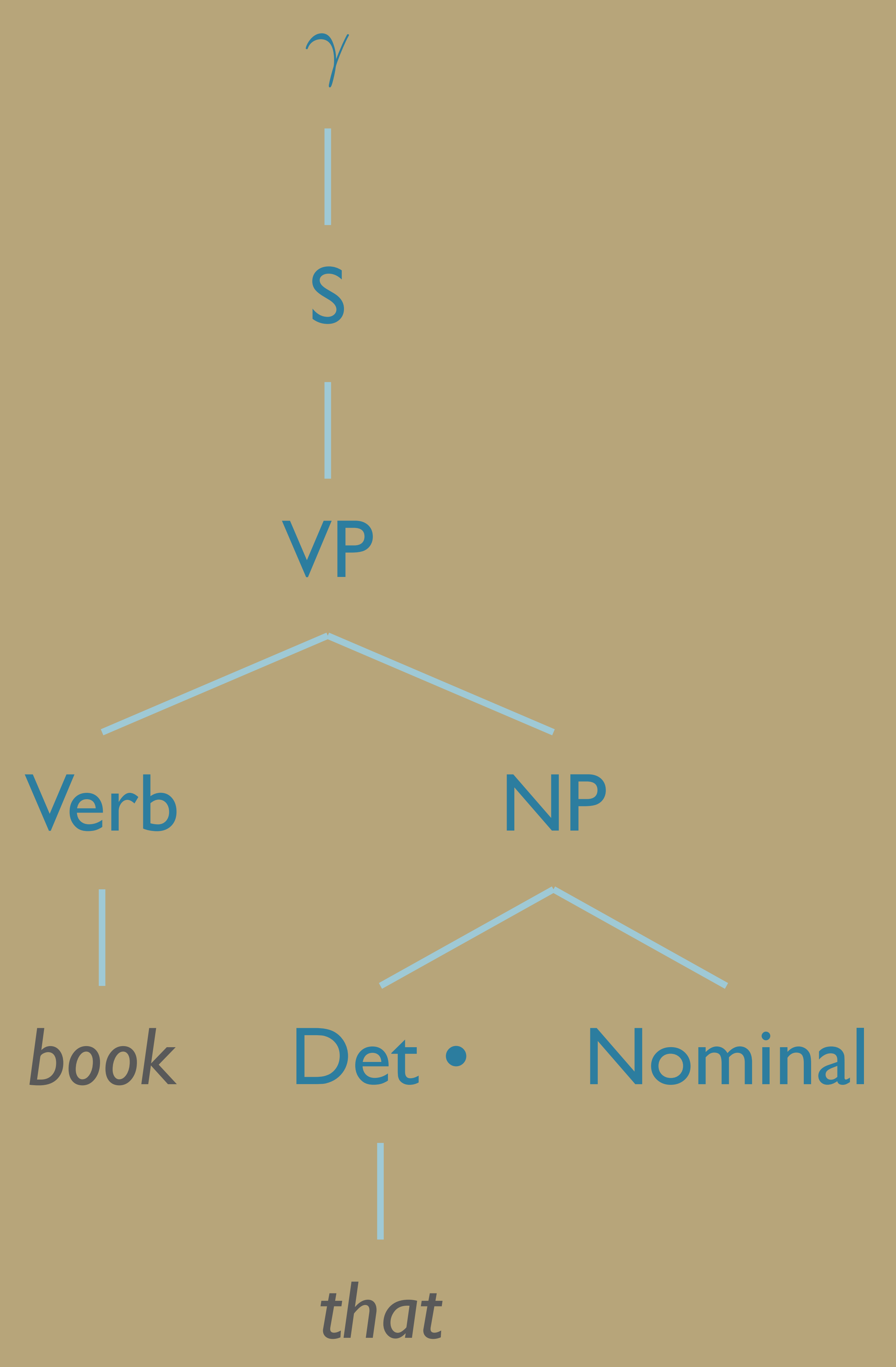
Book that flight

S0: γ → • S [0,0]
S3: S → VP • [0,1]
S8: VP → Verb • NP [0,1]
S21: NP → • Det Nominal [1,1]
S23: Det → “that” • [1,2]



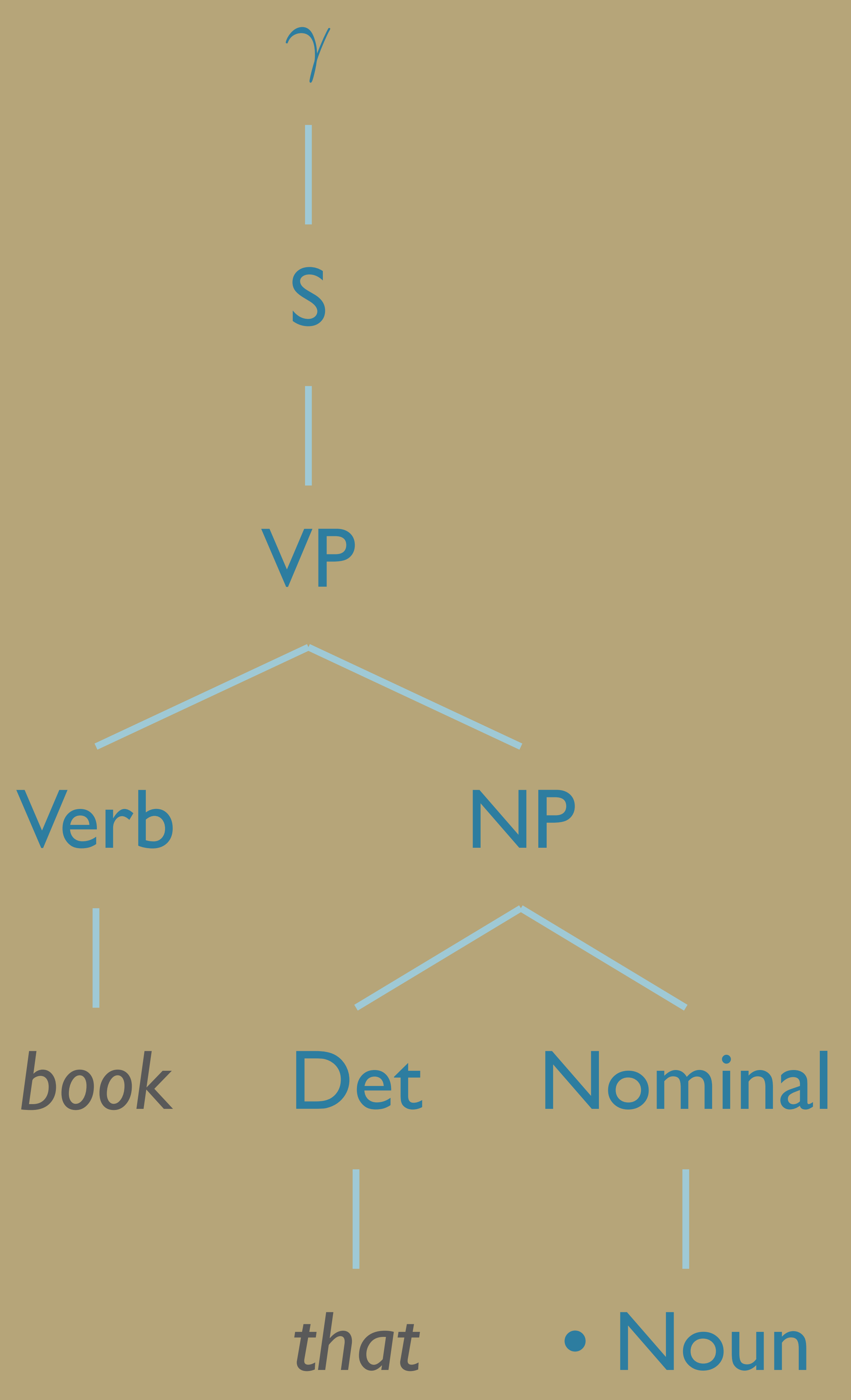
Book that flight

S0: $\gamma \rightarrow \cdot\ S\ [0,0]$
S3: $S \rightarrow VP\ \cdot\ [0,1]$
S8: $VP \rightarrow Verb\ \cdot\ NP\ [0,1]$
S21: $NP \rightarrow Det\ \cdot\ Nominal\ [1,2]$



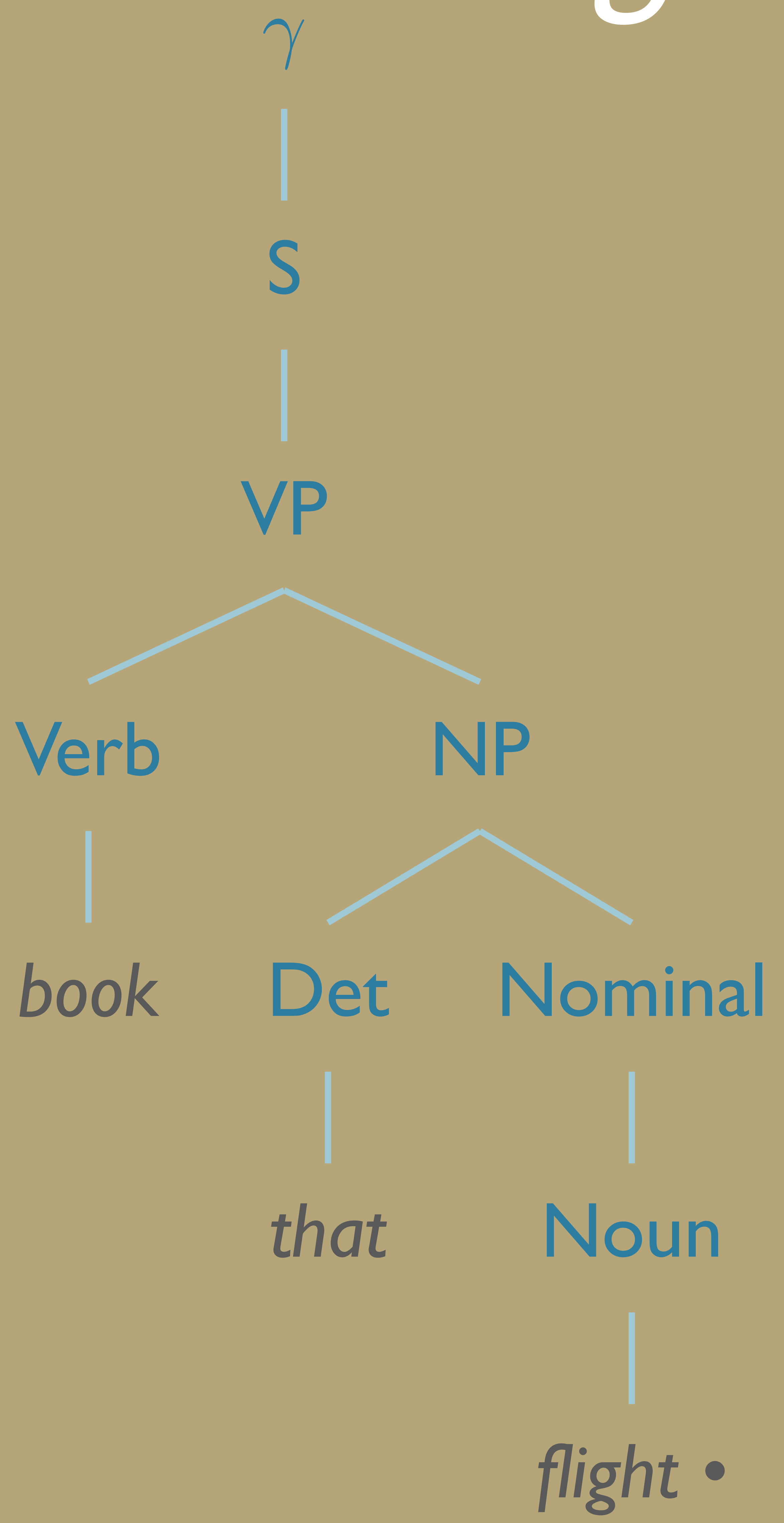
Book that flight

S0: $\gamma \rightarrow \bullet$ S [0,0]
S3: S $\rightarrow$ VP $\bullet$ [0,1]
S8: VP $\rightarrow$ Verb $\bullet$ NP [0,1]
S21: NP $\rightarrow$ Det $\bullet$ Nominal [1,2]
S25: Nominal $\rightarrow$ $\bullet$ Noun [2,2]



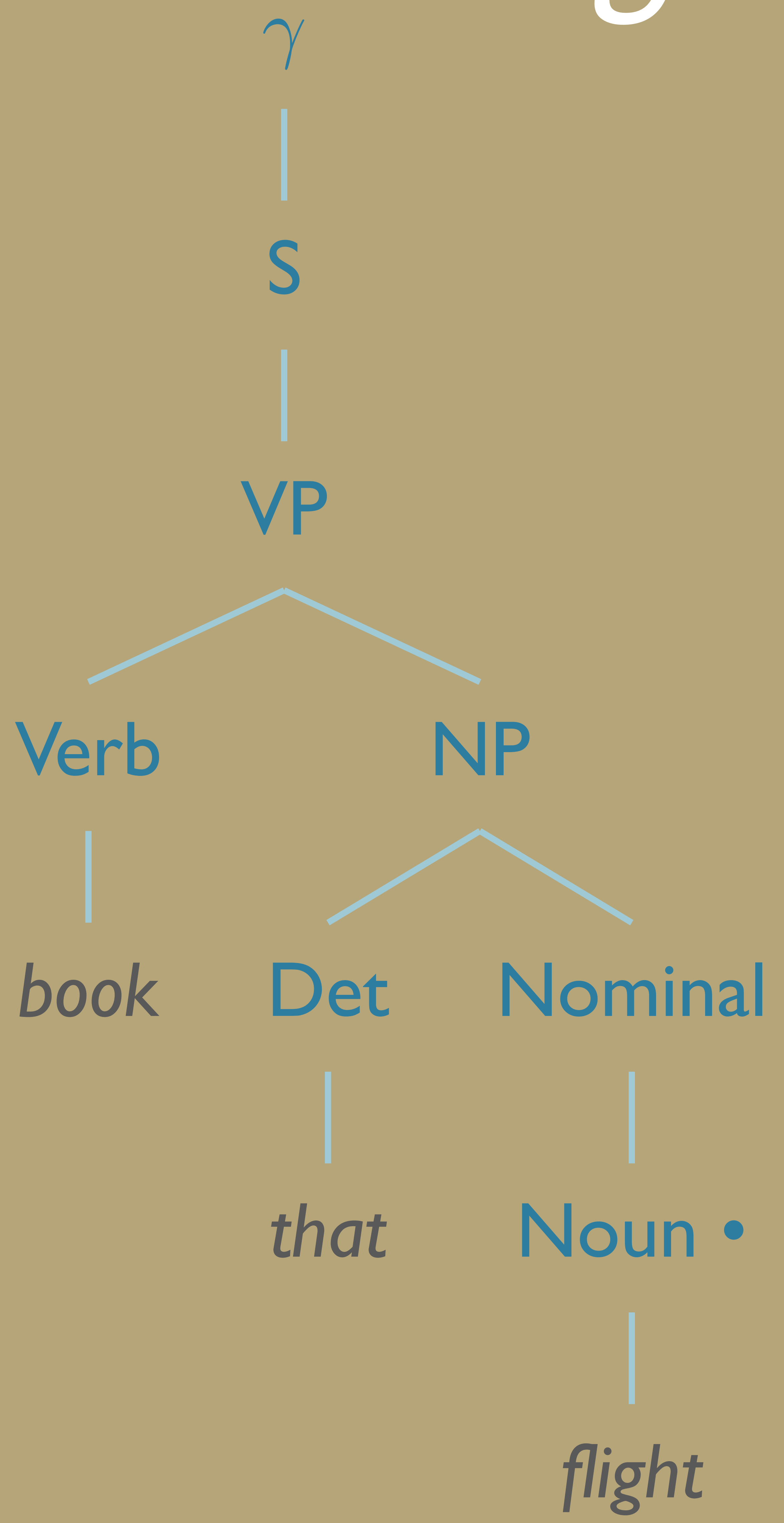
Book that flight

S0: γ → • S [0,0]
S3: S → VP • [0,1]
S8: VP → Verb • NP [0,1]
S21: NP → Det • Nominal [1,2]
S25: Nominal → • Noun [2,2]
S28: Noun → "flight" • [2,3]



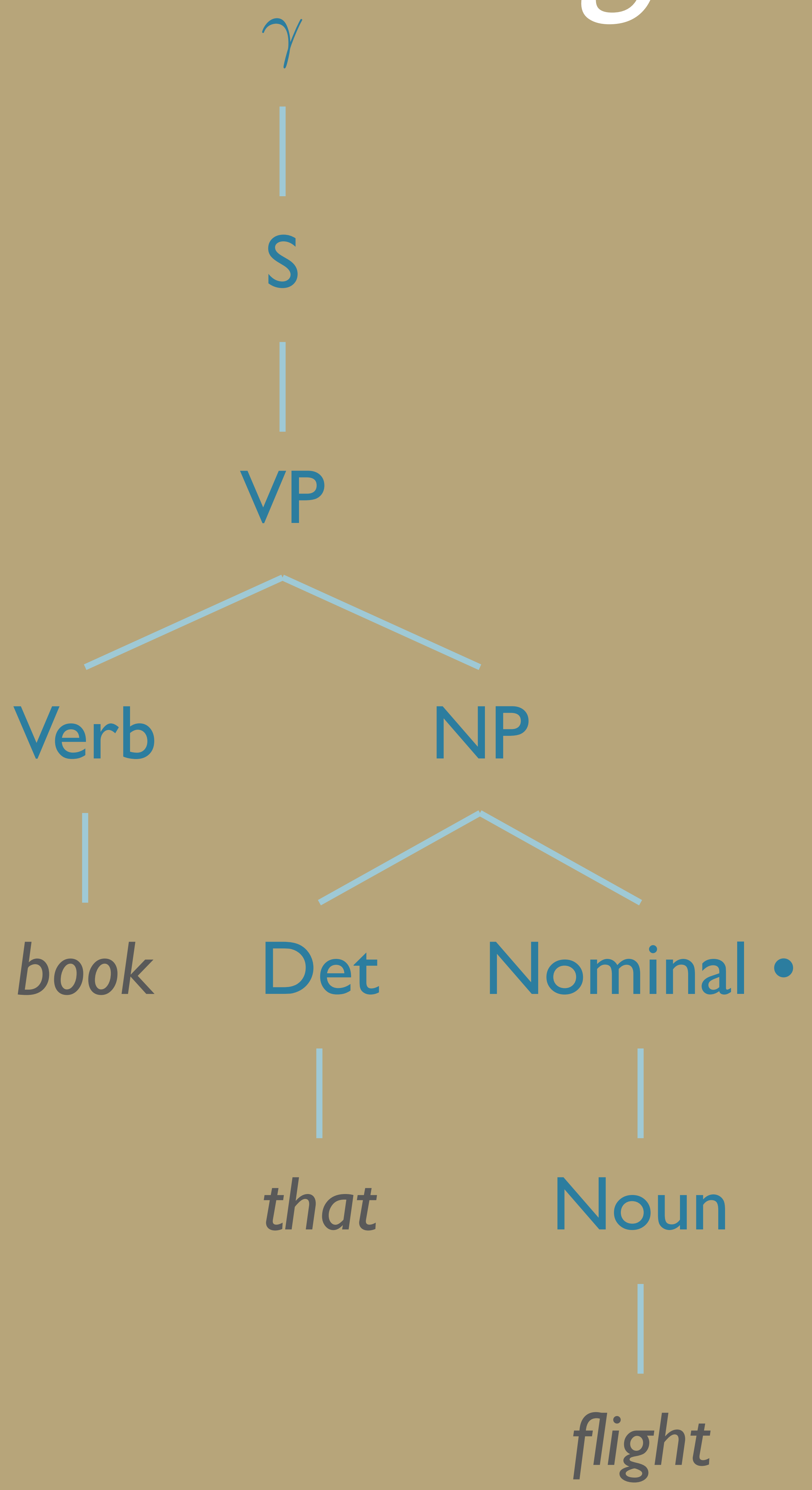
Book that flight

S0: γ → • S [0,0]
S3: S → VP • [0,1]
S8: VP → Verb • NP [0,1]
S21: NP → Det • Nominal [1,2]
S25: Nominal → Noun • [2,3]



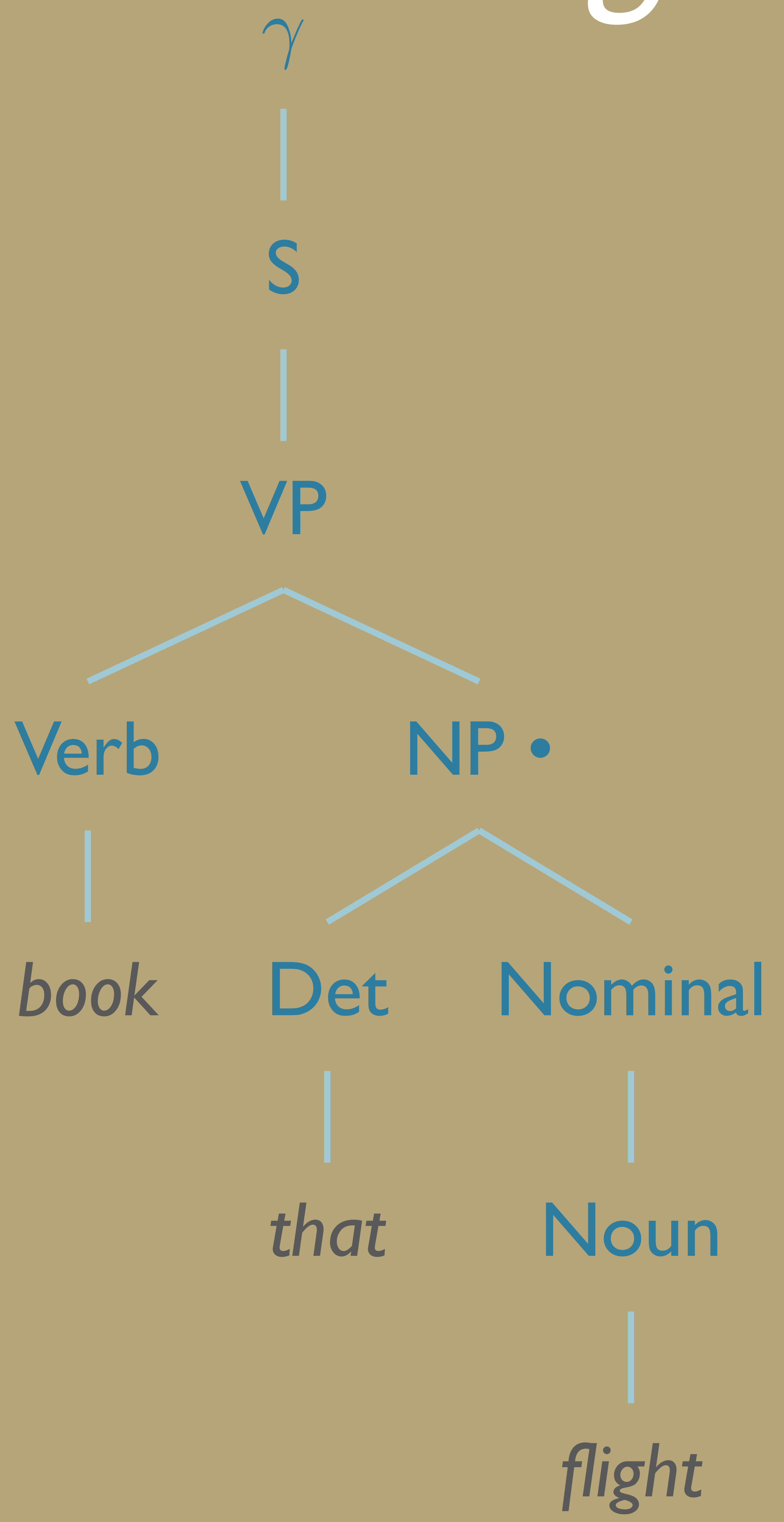
Book that flight

S0: γ → • S [0,0]
S3: S → VP • [0,1]
S8: VP → Verb • NP [0,1]
S21: NP → Det Nominal • [1,3]



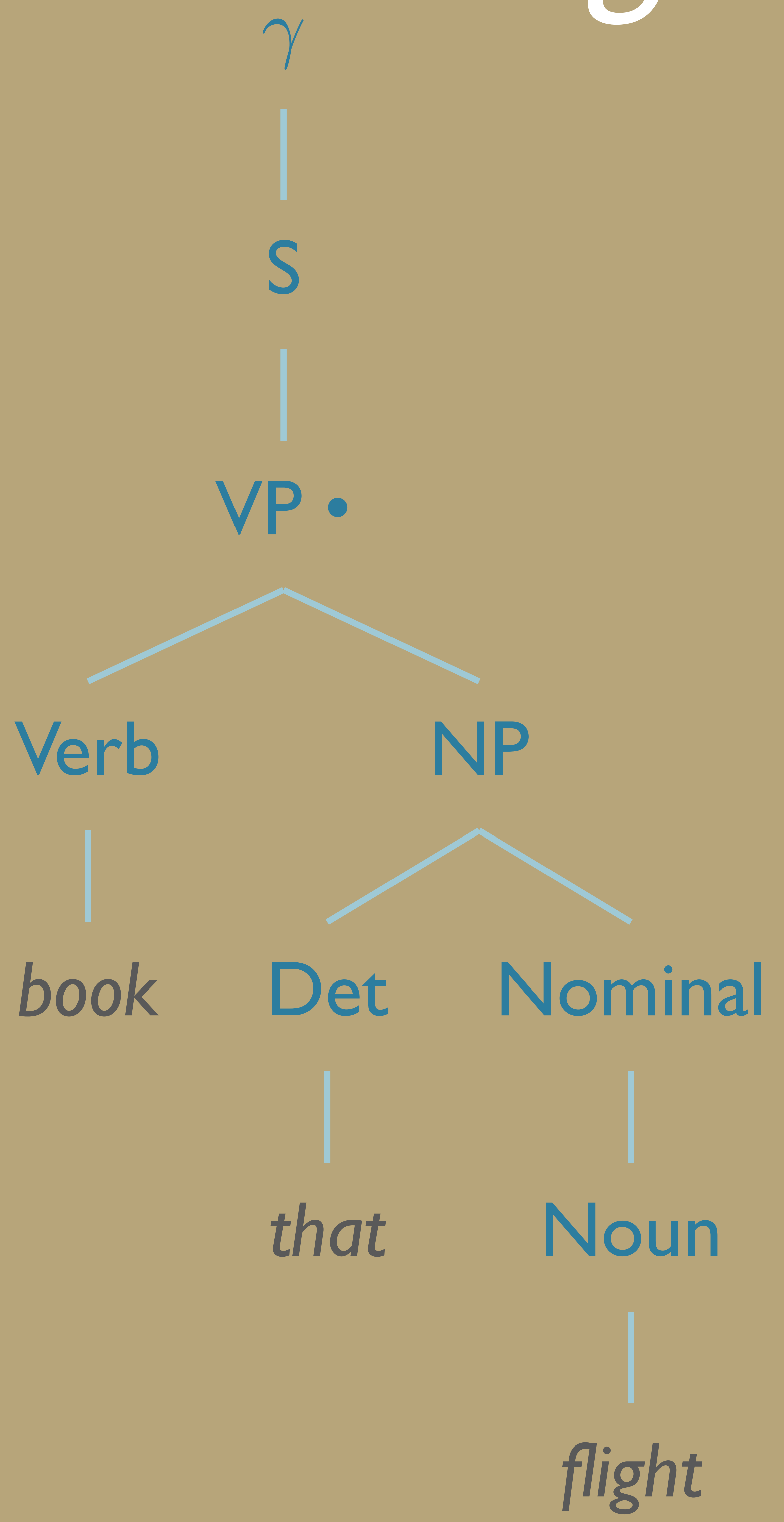
Book that flight

S0: γ → • S [0,0]
S3: S → VP • [0,1]
S8: VP → Verb NP • [0,3]



Book that flight

S0: γ → • S [0,0]
S3: S → VP • [0,3]

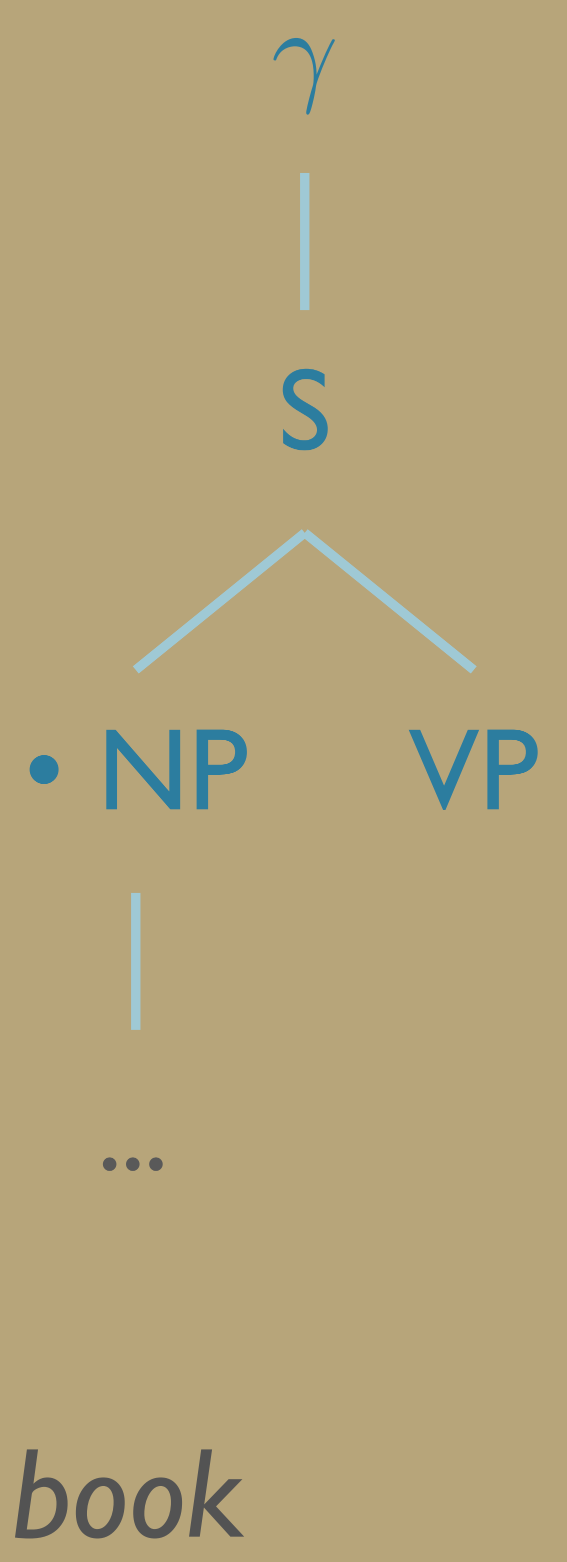


What About Dead Ends?

Book that flight

S0: $\gamma \rightarrow \bullet$ S [0,0]
S1: S $\rightarrow \bullet$ NP VP [0,0]

NP ~~$\rightarrow \bullet$ Pronoun~~
NP ~~$\rightarrow \bullet$ Proper-Noun~~
NP ~~$\rightarrow \bullet$ Det Nominal~~



What About Recursion?

What about recursion?

- We now have a top-down parser in hand. Does it enter infinite loops on rules like S -> S ‘and’ S?
- No!

procedure ENQUEUE(*state*, *chart-entry*)
  if *state* is not already in *chart-entry* **then**
    PUSH(*state*, *chart-entry*)
  end

Exercise: parse ‘table and chair’ using the very simple grammar
Nom -> Nom ‘and’ Nom | ‘table’ | ‘chair’

HW #3

CKY Parsing: Goals

- Complete implementation of CKY parser
- Implement dynamic programming approach
- Incorporate/follow backpointers to recover parse

Implementation

- Build full parser
- Can use any language, per course policies
- You may use existing data structures for rules, trees
  - e.g. NLTK has nice `tree` data structure
  - CKY algorithm must be your own
- Dynamic programming table filling crucial!
- Will use smaller grammar (similar to HW #1)
- Back to ATIS for HW #4

Implementation

- For CKY Implementation:
  - NLTK's **`CFG.productions()`** method:
    - optional `rhs=` argument *only looks at first token of RHS*

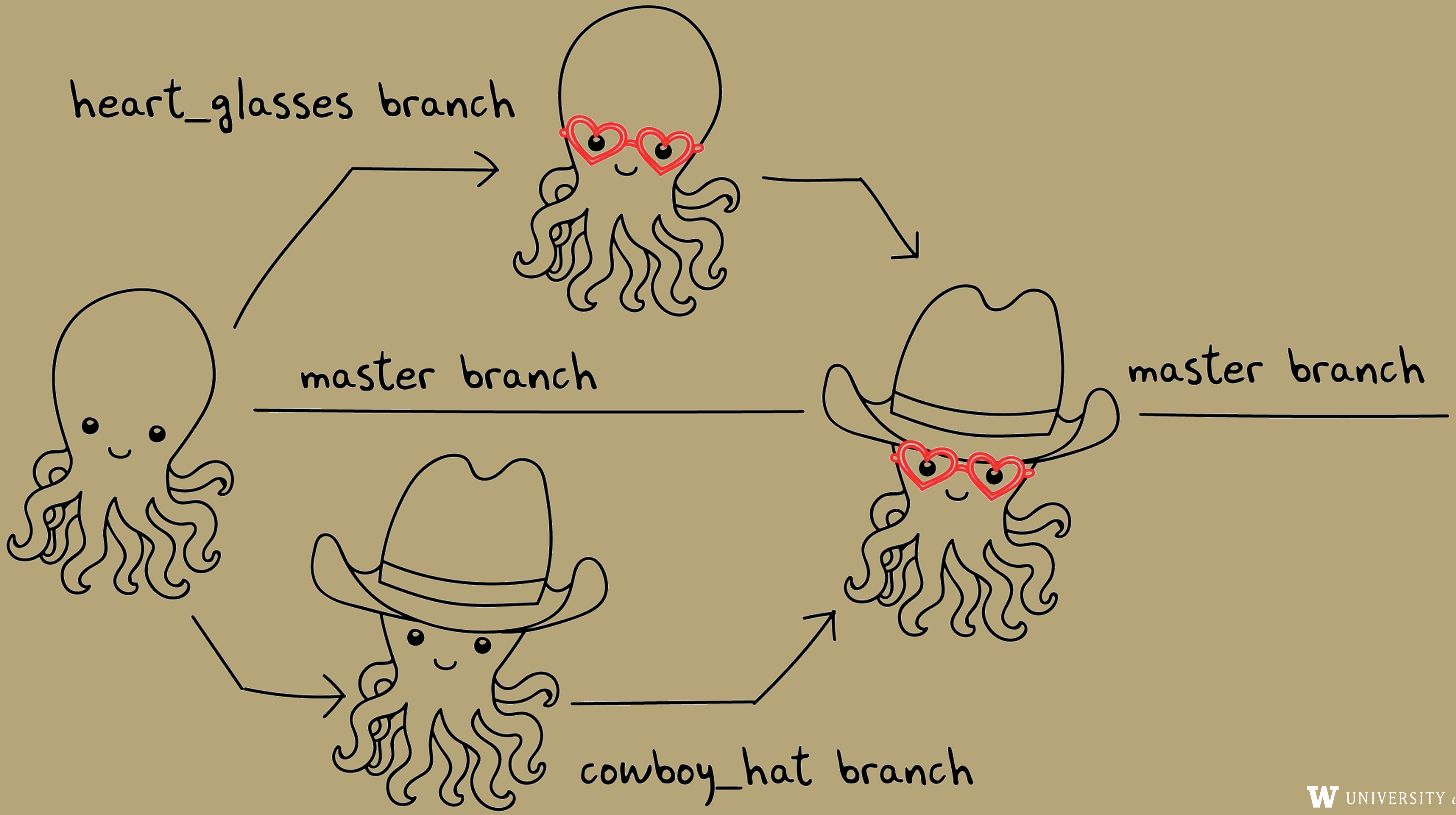
Notes

- Teams:
  - You may work in teams of two on this assignment
- Test grammar
  - Pre-converted to CNF
  - Start symbol: $\mathbf{TOP}$
  - Parse should span input and be rooted at: $\mathbf{TOP}$

Some Collaboration Basics

Git Branches

- Good for semi-isolating your development code from the shared, reviewed code



Recommended Git Flow

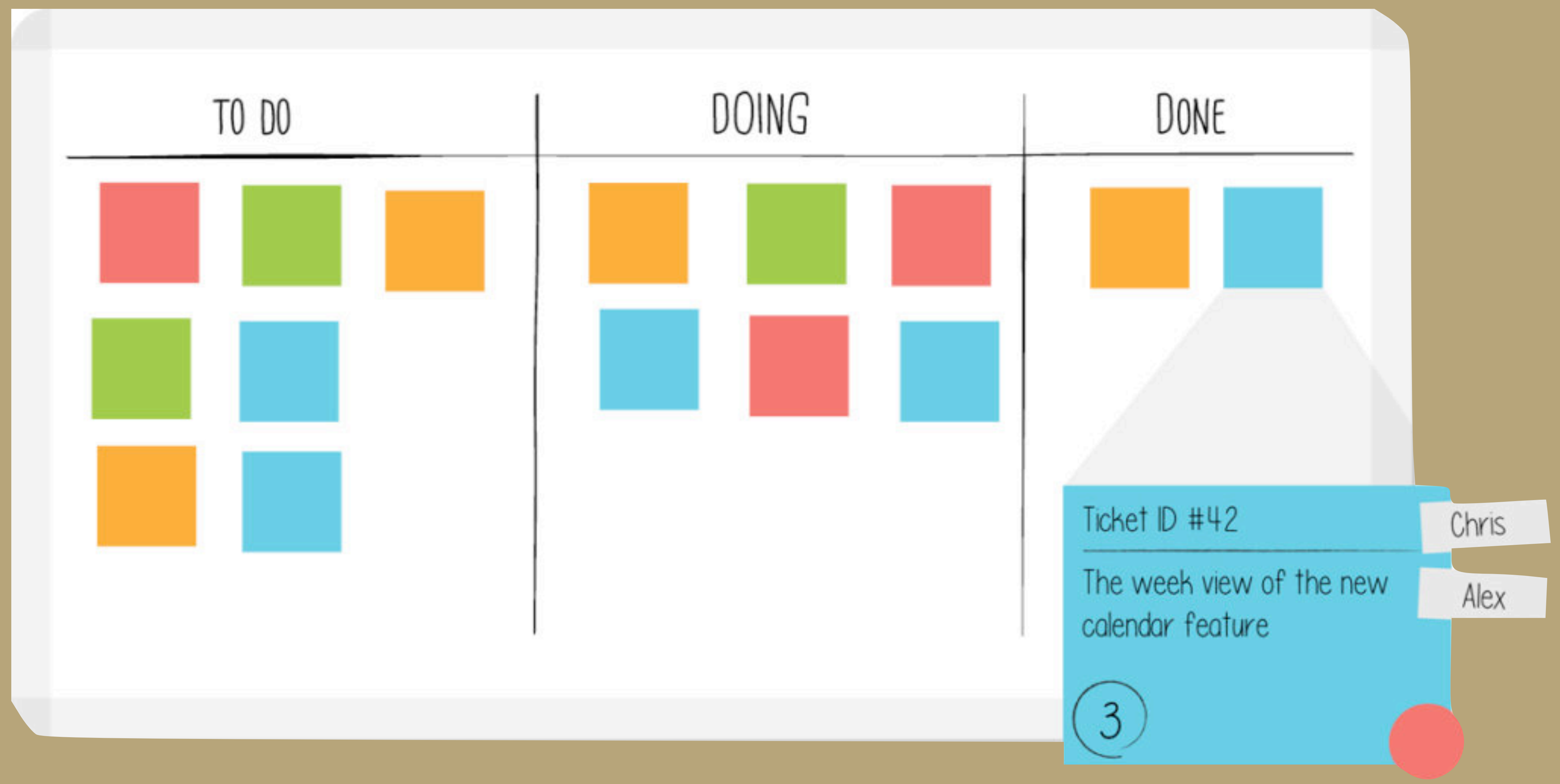
- Initialize a git repository, with a `master` branch
  - (Create initial checkin, if necessary)
- Create a new branch, maybe "`adding_rule_objects`"
- Make regular checkins on your branch (like saving)
- Switch to `master` branch, and "pull"
- Merge your branch to master
- …rinse & repeat
- If using GitHub (or GitLab, etc): **MUST BE PRIVATE REPO!**

Communication: Check-ins

- For check-ins, three main points:
  - What have you been working on?
  - What do you plan to work on next?
  - Is there anything “blocking” you?
- In industry, these brief check-ins among small teams are often done daily

Project Planning: Kanban Boards

- Before you start working:
  - Write out tasks on sticky notes.
  - Place in three columns:
    - To-Do
    - Doing
    - Done
  - As you work, you can move them from column to column
  - Add tasks as new issues come up



- trello.com – has free online implementation of Kanban Boards