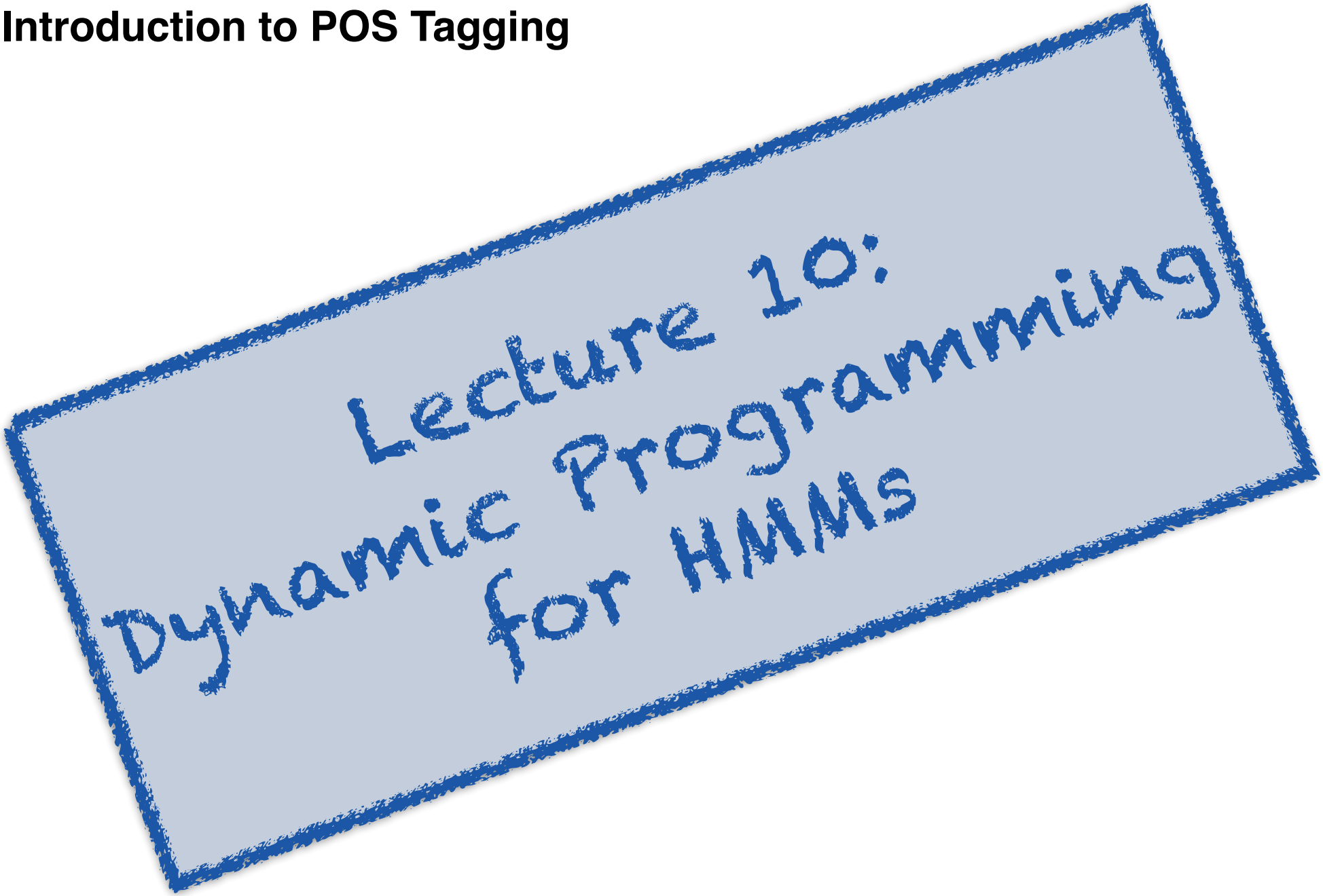
Lecture 10:
Introduction to POS Tagging



HMM decoding (Viterbi)

We are given a sentence $\mathbf{w} = w^{(1)} \ldots w^{(N)}$

$\mathbf{w}$= *"she promised to back the bill"*

We want to use an HMM tagger to find its POS tags $\mathbf{t}$

$$\mathbf{t}^* = \text{argmax}_{\mathbf{t}} \, P(\mathbf{w}, \mathbf{t})$$

$$= \text{argmax}_{\mathbf{t}} \, P(t^{(1)}) \cdot P(w^{(1)} | \, t^{(1)}) \cdot P(t^{(2)} | \, t^{(1)}) \cdot \ldots \cdot P(w^{(N)} | \, t^{(N)})$$

But: with T tags, ***w*** *has* $O(T^N)$ *possible tag sequences!*

To do this efficiently (in $O(T^2N)$ time), we will use a **dynamic programming** technique called the **Viterbi algorithm** which exploits the **independence assumptions** in the HMM.

Dynamic programming

Dynamic programming is a general technique to solve certain complex search problems by memoization

1.) Recursively decompose the large search problem into smaller **subproblems** that can be solved efficiently

- There is only a polynomial number of subproblems.

2.) Store (memoize) the solutions of each subproblem in a **common data structure**

- Processing this data structure takes polynomial time

The Viterbi algorithm

A dynamic programming algorithm which finds the best (=most probable) tag sequence $\mathbf{t}^*$ for an input sentence $\mathbf{w}$: $\mathbf{t}^* = \text{argmax}_{\mathbf{t}} P(\mathbf{w} \mid \mathbf{t})P(\mathbf{t})$

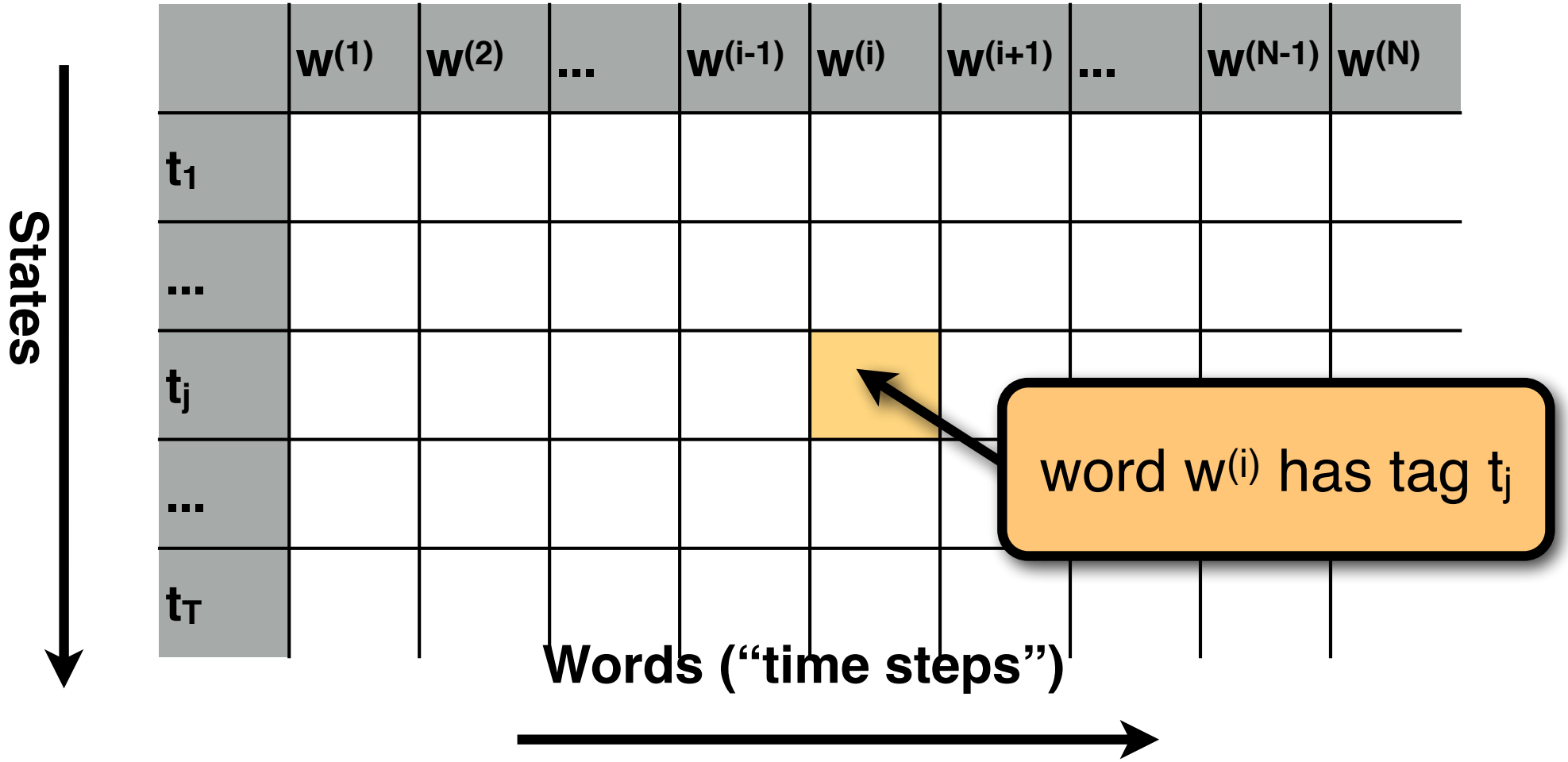
Complexity: linear in the sentence length.

With a bigram HMM, Viterbi runs in $O(T^2N)$ steps for an input sentence with N words and a tag set of T tags.

The independence assumptions of the HMM tell us how to break up the big search problem (find $\mathbf{t}^* = \text{argmax}_{\mathbf{t}} P(\mathbf{w} \mid \mathbf{t})P(\mathbf{t})$) into smaller subproblems.

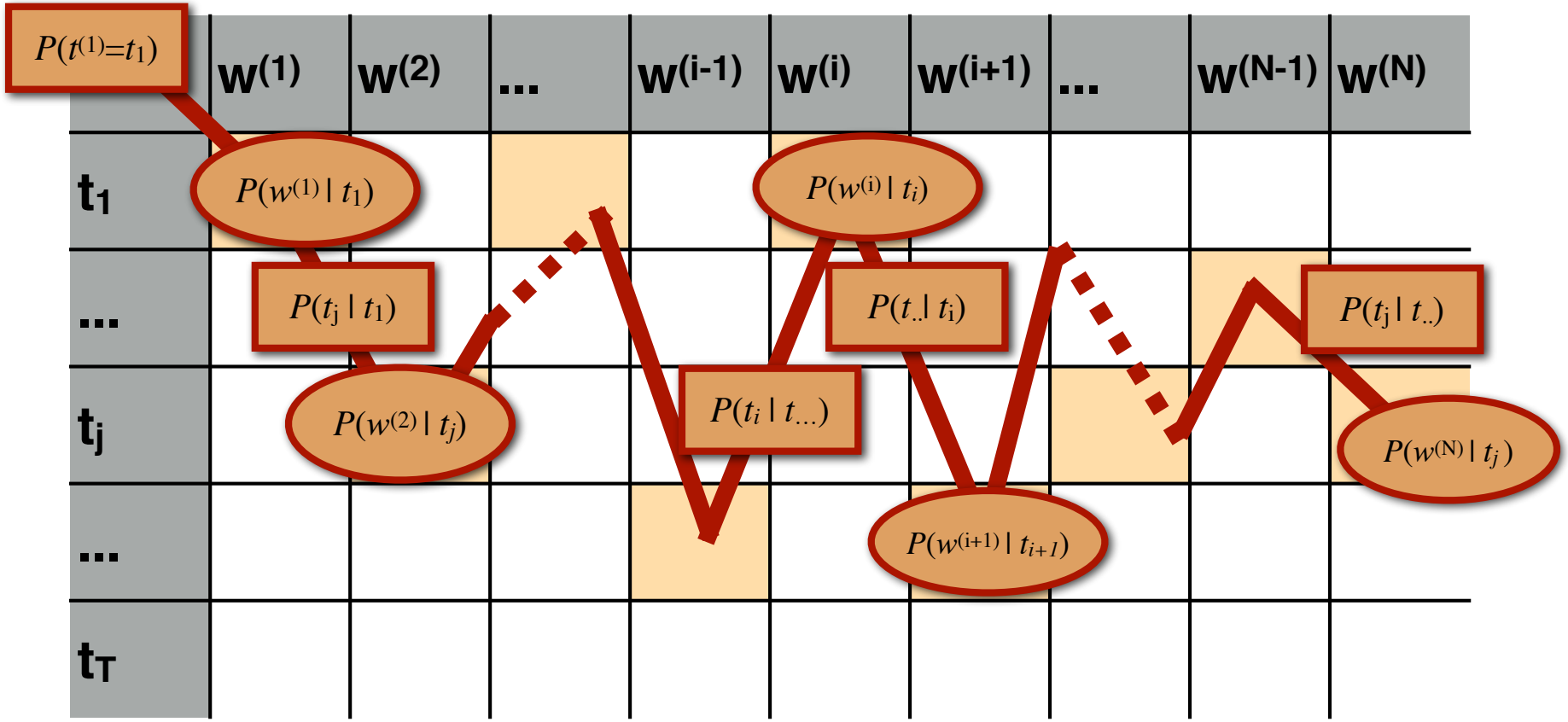
The data structure used to store the solution of these subproblems is the trellis.

Bookkeeping: the trellis



We use a N×T table ("**trellis**") to keep track of the HMM.
The HMM can assign one of the T tags to each of the N words.

Computing $P(\mathbf{t},\mathbf{w})$ for one tag sequence



One path through the trellis = one tag sequence

Viterbi: Basic Idea

Task: Find the tag sequence $t^{(1)} \ldots t^{(N-1)} t^{(N)}$
that maximizes the joint probability

$$\pi\left(t^{(1)}\right) P\left(w^{(1)} \mid t^{(1)}\right) \prod_{i=2}^{N} P\left(t^{(i)} \mid t^{(i-1)}\right) P\left(w^{(i)} \mid t^{(i)}\right)$$

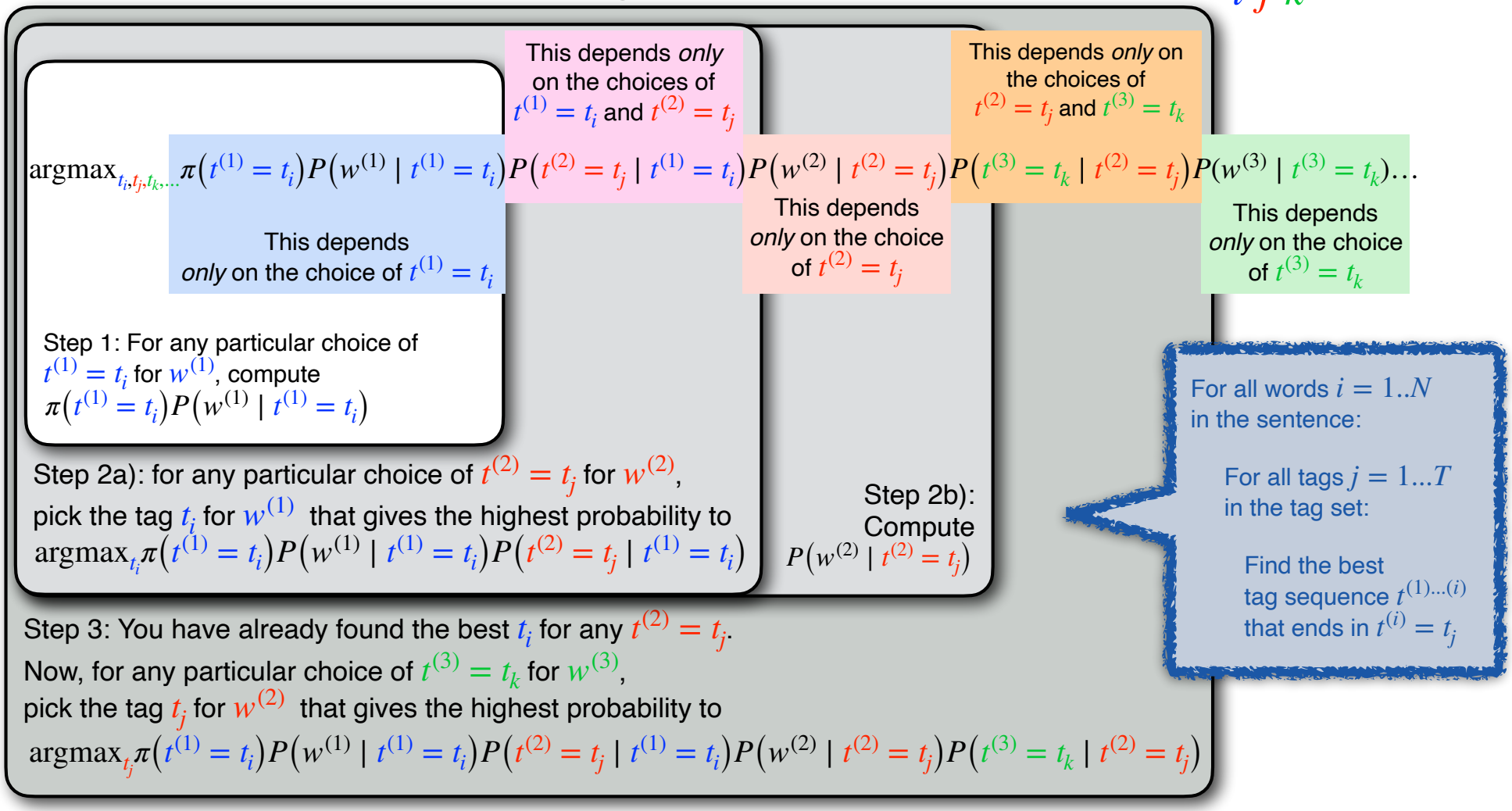
The choice of $t^{(1)}$ affects the probability of $t^{(2)}$,
which in turn affects the probability of $t^{(3)}$, etc:

$$\pi\left(t^{(1)}\right) P\left(w^{(1)} \mid t^{(1)}\right) P\left(t^{(2)} \mid t^{(1)}\right) P\left(w^{(2)} \mid t^{(2)}\right) P\left(t^{(3)} \mid t^{(2)}\right) \ldots$$

→ We **cannot fix** $t^{(1)}$ (or any tag) until the **end of the sentence**!

Exploiting the independence assumptions

You want to find the best tag sequence $t^{(1)}t^{(2)}t^{(3)}\ldots = t_i t_j t_k \ldots$



Viterbi: Basic Idea

Assume we knew (for any tag t_j) **the maximum probability** of any *complete* sequence $t^{(1)} \ldots t^{(N)}$ that ***ends* in that tag** $t^{(N)} = t_j$ [*N*: last word in **w**]

Call that probability the **Viterbi probability** of tag t_j at position N, and store it as **trellis[N][j].viterbi**

Then, the **probability of the *best* tag sequence** (i.e. the maximum probability of *any* complete sequence $t^{(1)} \ldots t^{(N)}$) **for our sentence** is

$\mathbf{max}_{k \in \{1,..,T\}}(\mathbf{trellis}[N][k].\mathbf{viterbi})$

Viterbi: Basic Idea

Viterbi probability of tag t_j for word $w^{(i)}$: **trellis[i][j].viterbi**

The highest probability $P\left(\mathbf{w}^{(1)\ldots(i)}, \mathbf{t}^{(1)\ldots(i)}\right)$ of the prefix $\mathbf{w}^{(1)\ldots(i)}$ and *any* tag sequence $\mathbf{t}^{(1)\ldots(i)}$ ending in $t^{(i)} = t_j$

$$\text{trellis[i][j].viterbi} = \max P(\text{w}^{(1)}\ldots\text{w}^{(\text{i})}, \text{t}^{(1)}\ldots, \text{t}^{(\text{i})} = \text{t}_\text{j})$$

The probability of the best tag sequence overall is given by:

$\max_k$ trellis[N][k].viterbi

(the largest entry in the last column of the trellis)

The Viterbi probability trellis[i][j].viterbi (for any cell in the trellis) can easily be computed based on the cells in the preceding column, trellis[i-1][k].viterbi

Viterbi: Basic Idea

Viterbi probability of tag t_j for word $w^{(i)}$: **trellis[i][j].viterbi**

The highest probability $P\left(\mathbf{w}^{(1)\ldots(i)}, \mathbf{t}^{(1)\ldots(i)}\right)$ of the prefix $\mathbf{w}^{(1)\ldots(i)}$ and *any* tag sequence $\mathbf{t}^{(1)\ldots(i)}$ ending in $t^{(i)} = t_j$

Base case: First word $w^{(1)}$ in the sentence

$$\text{trellis}[1][j].\text{viterbi} = \pi(t_j) P(w^{(1)} \mid t_j)$$

Initial probability for tag t_j
Emission probability for $w^{(1)}$

Recurrence: Any other word $w^{(i)}$ in the sentence

$$\text{trellis}[i][j].\text{viterbi} = \max_k \left(\text{trellis}[i-1][k].\text{viterbi} \times P(t_j \mid t_k) P(w^{(i)} \mid t_j) \right)$$

Viterbi probability of tag t_k for the preceding word $w^{(i-1)}$

Transition prob. for t_j given t_k

Emission prob. for $w^{(i)}$ given t_j

Initialization

For a bigram HMM:

Given an N-word sentence $w^{(1)} \ldots w^{(N)}$ and a tag set consisting of T tags, create a trellis of size N×T

In the first column, initialize each cell $\text{trellis}[1][k]$ as

$$\text{trellis}[1][k] := \pi(t_k) P(w^{(1)} \mid t_k)$$

(there is only a single tag sequence for the first word that assigns a particular tag to that word)

Viterbi: filling in the first column

	$w^{(1)}$
DT	$\pi(\text{DT}) \times P(w^{(1)} \mid \text{DT})$
...	
NNS	$\pi(\text{NNS}) \times P(w^{(1)} \mid \text{NNS})$
...	
VBZ	$\pi(\text{VBZ}) \times P(w^{(1)} \mid \text{VBZ})$

$\pi(\text{DT})$: probability that a sentence starts with DT

$P(w^{(1)} \mid \text{DT})$: probability that tag DT emits word $w^{(1)}$

We want to find the best (most likely) tag sequence for the entire sentence.

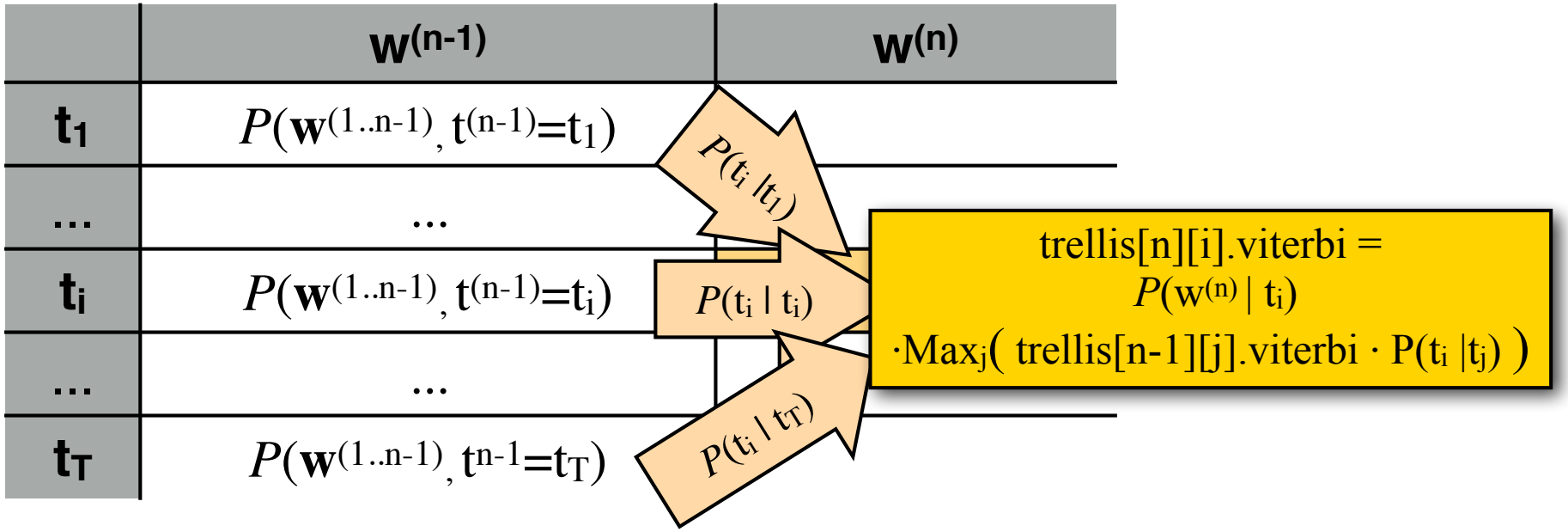
Each cell **trellis[i][j]** (corresponding to word $w^{(i)}$ with tag t_j) contains:

— **trellis[i][j].viterbi:** the probability of the best sequence ending in t_j

— **trellis[i][j].backpointer:** to the cell k in the previous column that corresponds to the best tag sequence ending in t_j

At any internal cell

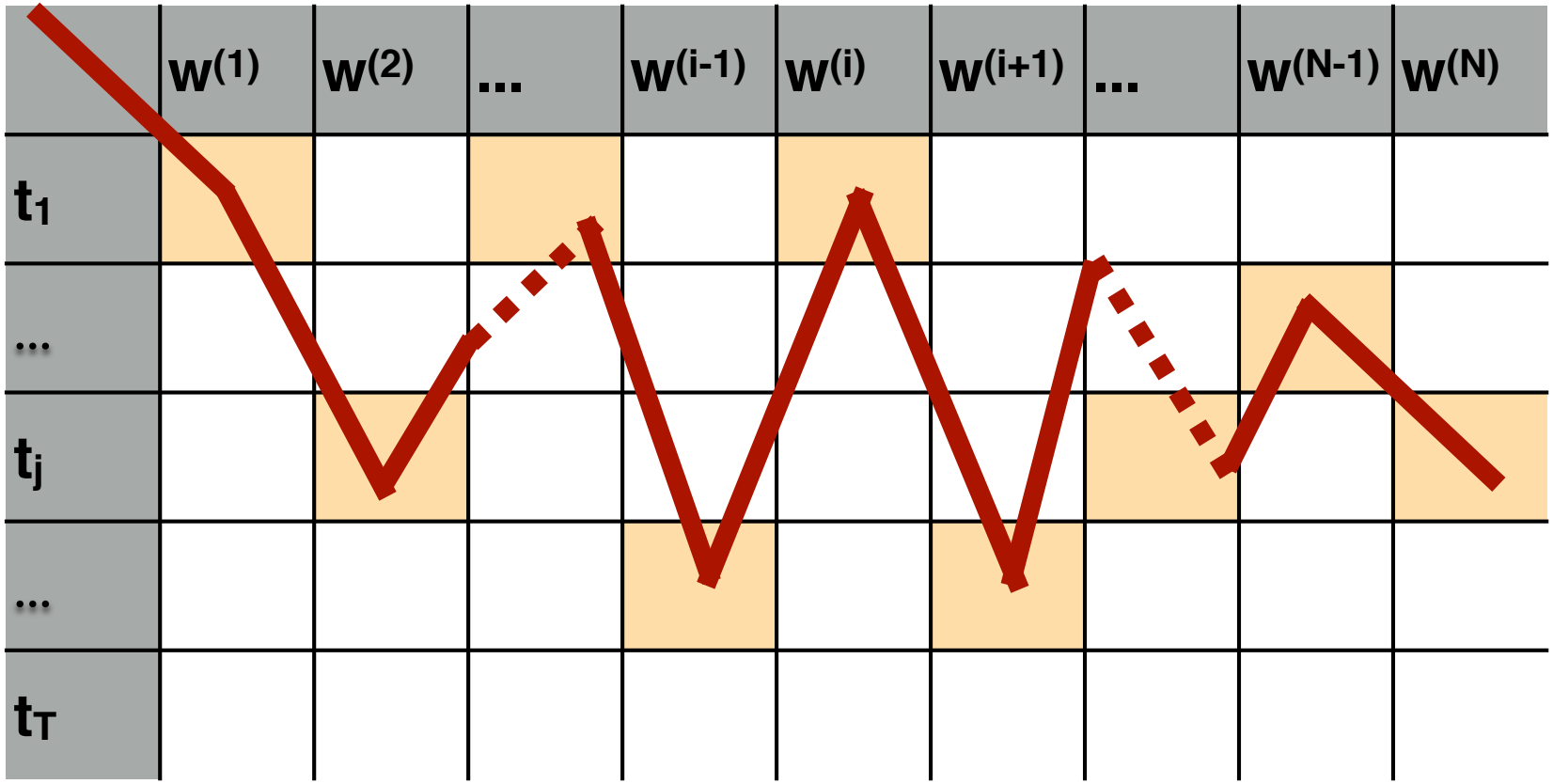
- For each cell in the preceding column: multiply its Viterbi probability with the transition probability to the current cell.
- Keep a single backpointer to the best (highest scoring) cell in the preceding column
- Multiply this score with the emission probability of the current word



At the end of the sentence

In the last column (i.e. at the end of the sentence) pick the cell with the highest entry, and trace back the backpointers to the first word in the sentence.

Retrieving $\mathbf{t}^* = \text{argmax}_{\mathbf{t}} P(\mathbf{t},\mathbf{w})$



By keeping **one backpointer** from each cell to the cell in the previous column that yields the highest probability, we can retrieve the most likely tag sequence when we're done.

Viterbi

Each cell trellis[i][j] (word $w^{(i)}$ with tag t_j) **contains**:

— **The Viterbi probability** trellis[i][j].viterbi:
The maximum probability $P(w^{(1)}\ldots w^{(i)}, t^{(1)},\ldots, t^{(i)} = t_j)$
of any tag sequence that ends in t_j for the prefix $w^{(1)\ldots(i)}$

— **A backpointer** trellis[i][j].backpointer = k*
to the cell trellis[i–1][k*] in the preceding column
that corresponds to the tag

To fill trellis[i][j], find the best cell in the previous column (trellis[i–1][k*])
based on the previous column and the transition probabilities $P(t_j \mid t_k)$

$$\text{k* for trellis[i][j]} := \text{Max}_k\ (\text{trellis[i–1][k]} \cdot P(t_j \mid t_k))$$

The entry in trellis[i][j] includes the emission probability $P(w^{(i)} \mid t_j)$

$$\text{trellis[i][j]} := P(w^{(i)} \mid t_j) \cdot (\text{trellis[i–1][k*]} \cdot P(t_j \mid t_{k^*}))$$

We also associate a backpointer from trellis[i][j] to trellis[i–1][k*]
Finally, return the highest scoring entry in the last column of the trellis
(= for the last word) and follow its backpointers

Viterbi

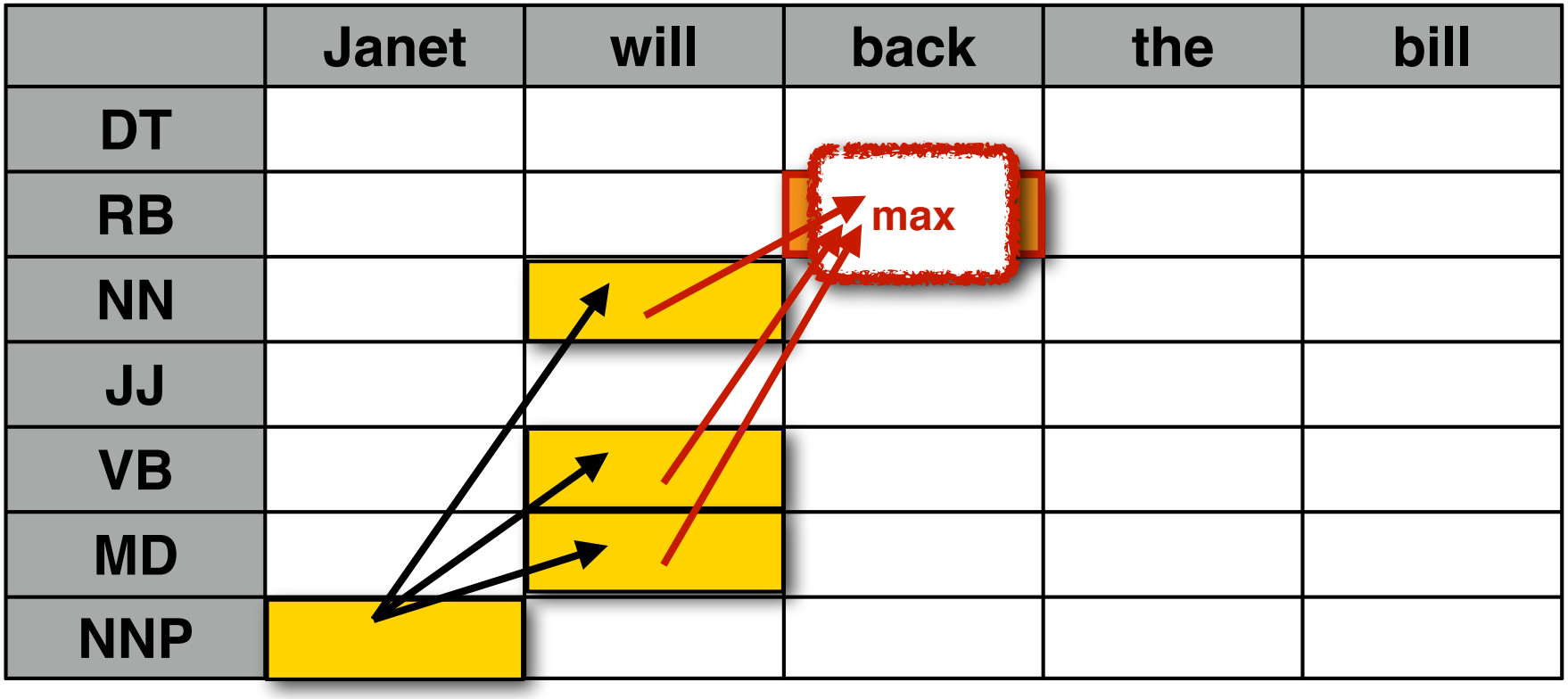
trellis[i][j].viterbi (word $w^{(j)}$, tag t_j) stores the probability of the best tag sequence for $w^{(1)} \ldots w^{(i)}$ that ends in t_j

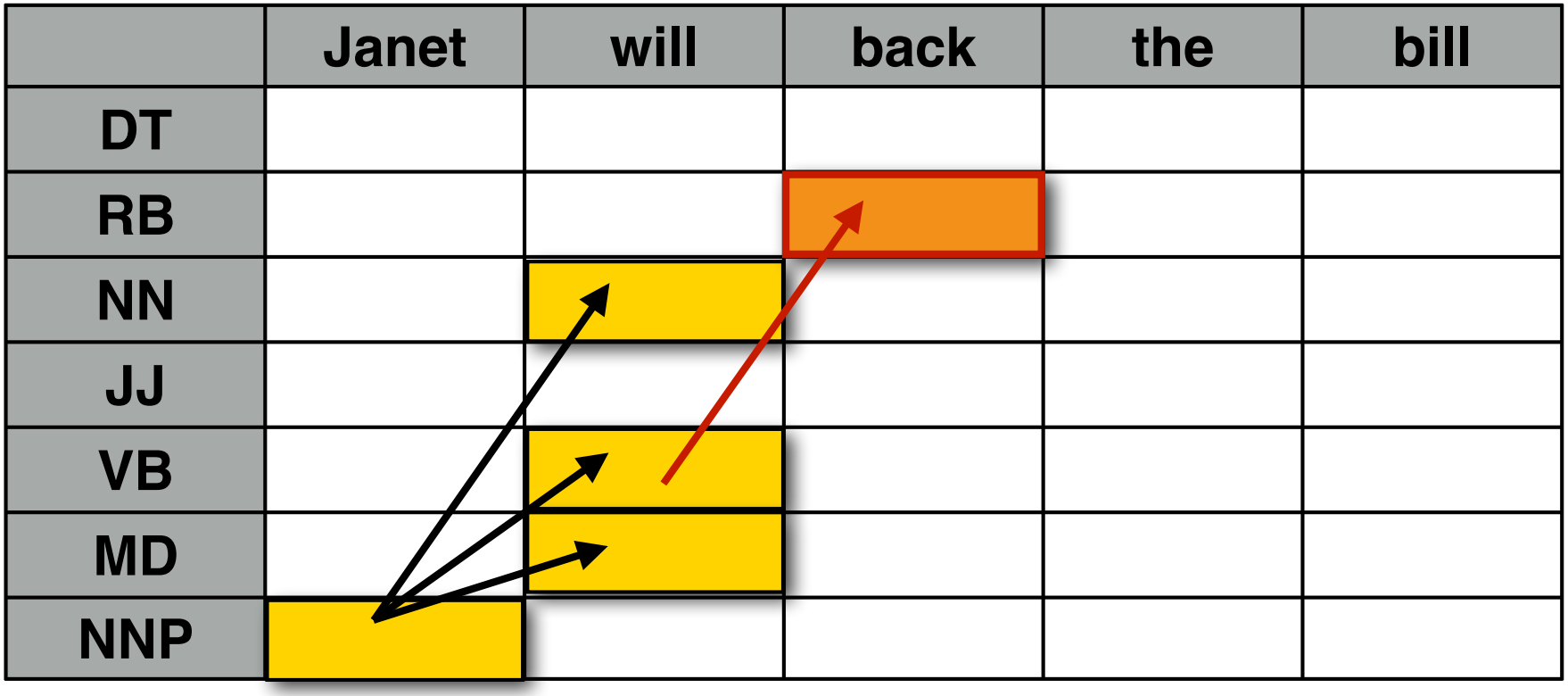
$$\text{trellis[i][j].viterbi} = \max P(w^{(1)} \ldots w^{(i)}, t^{(1)} \ldots, t^{(i)} = t_j)$$

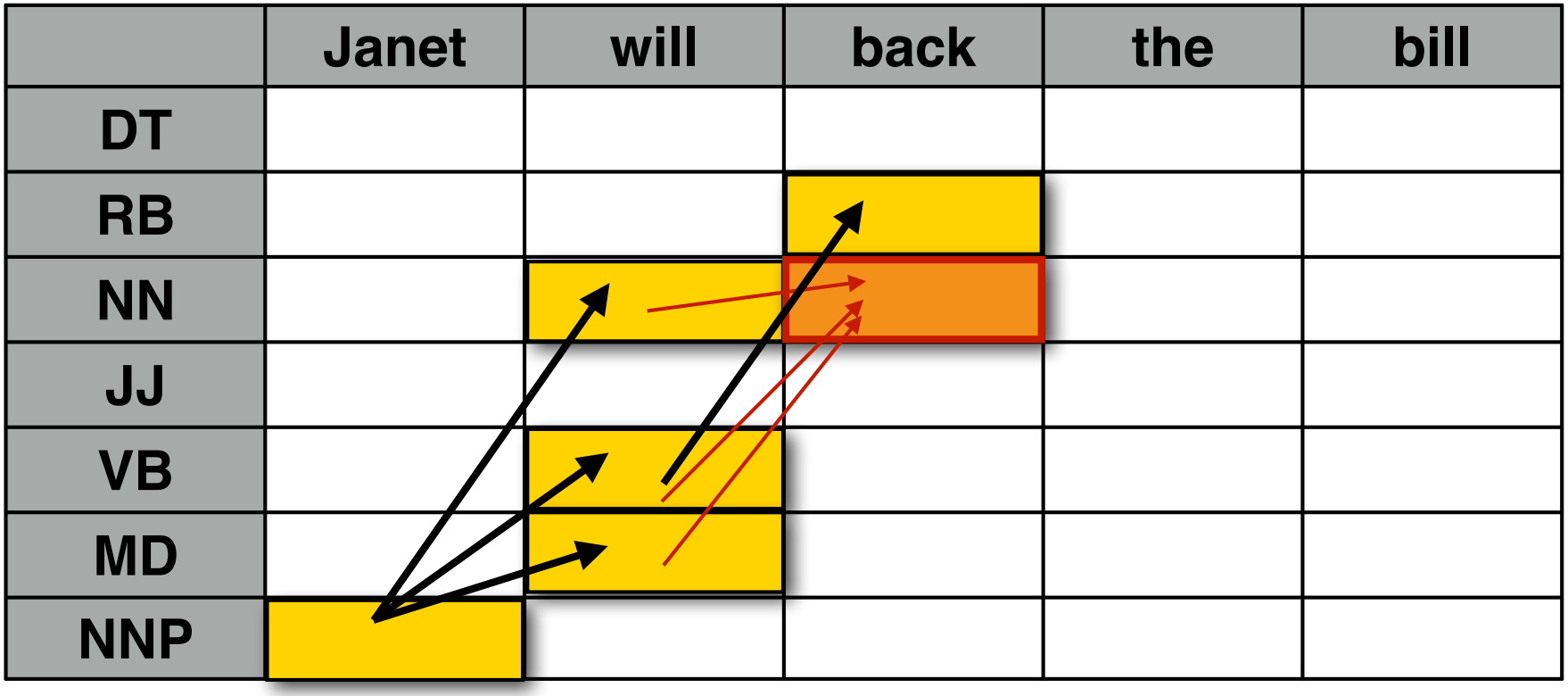
We can recursively compute trellis[i][j].viterbi from the entries in the previous column trellis[i-1][j].viterbi

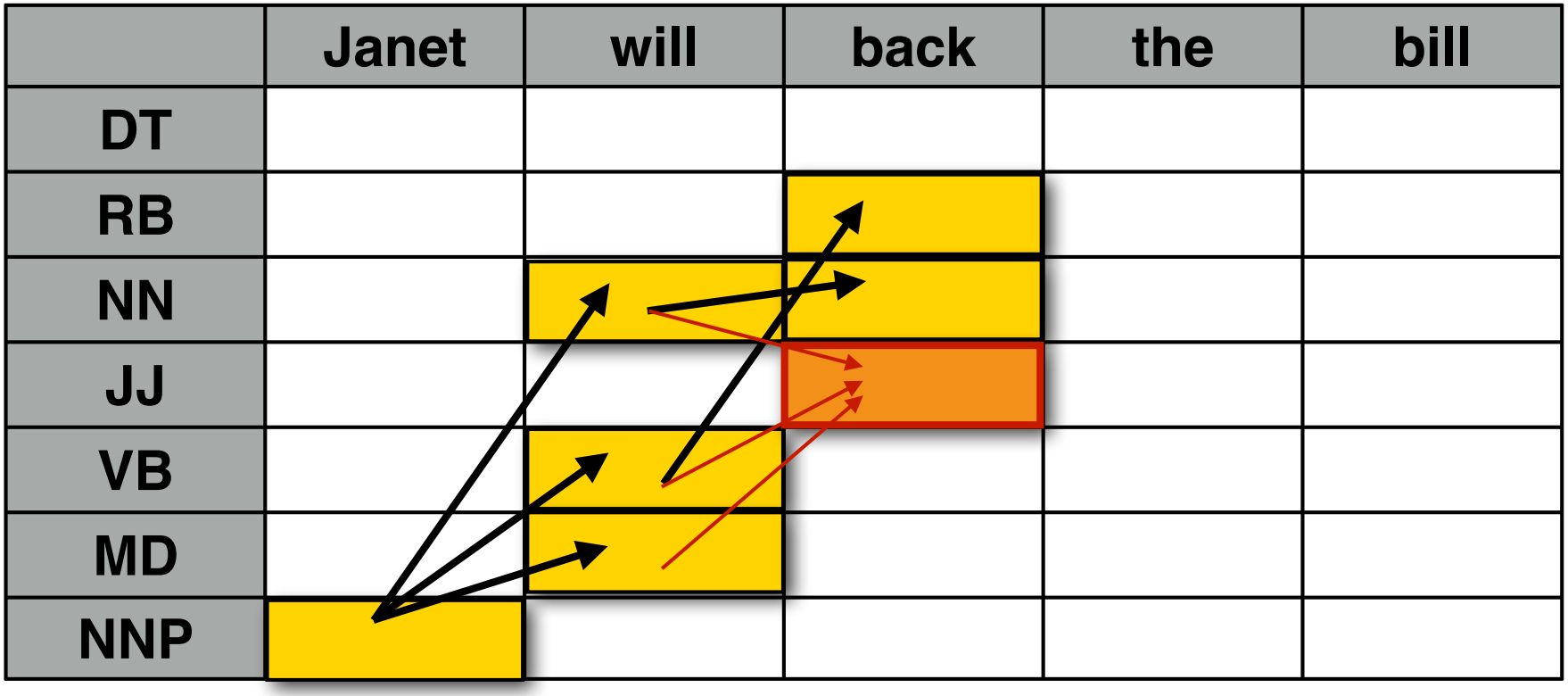
$$\text{trellis[i][j].viterbi} = P(w^{(i)} | t_j) \cdot \text{Max}_k(\text{ trellis[i-1][k].viterbi} P(t_j | t_k))$$

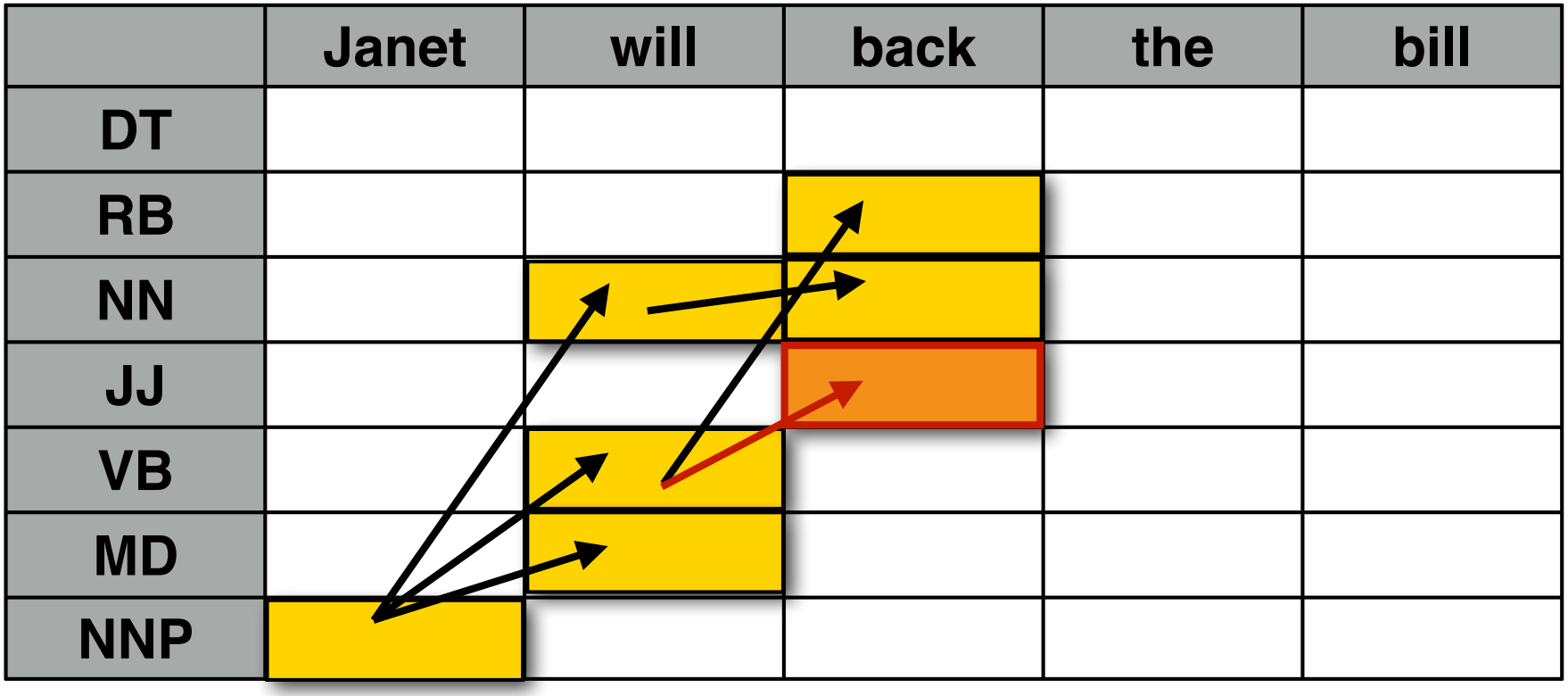
At the end of the sentence, we pick the highest scoring entry in the last column of the trellis

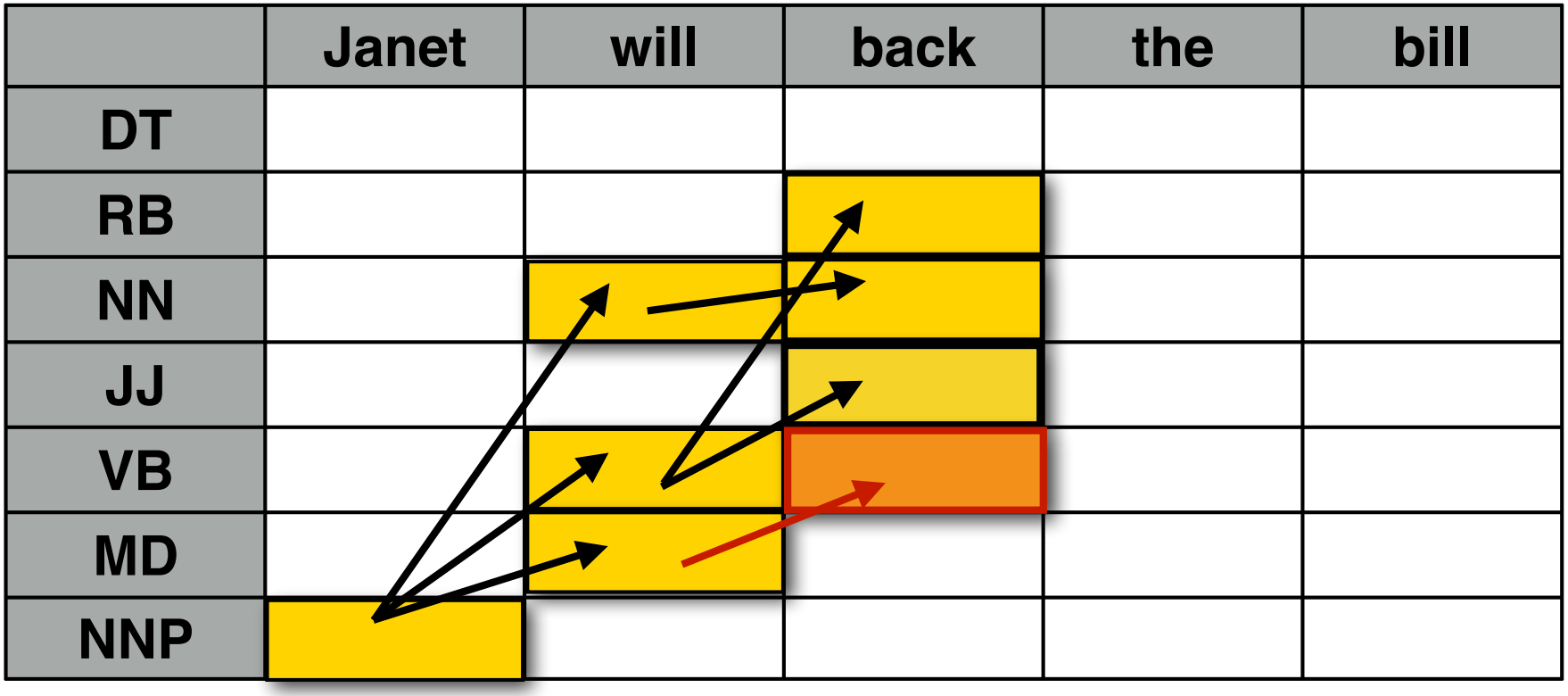

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max

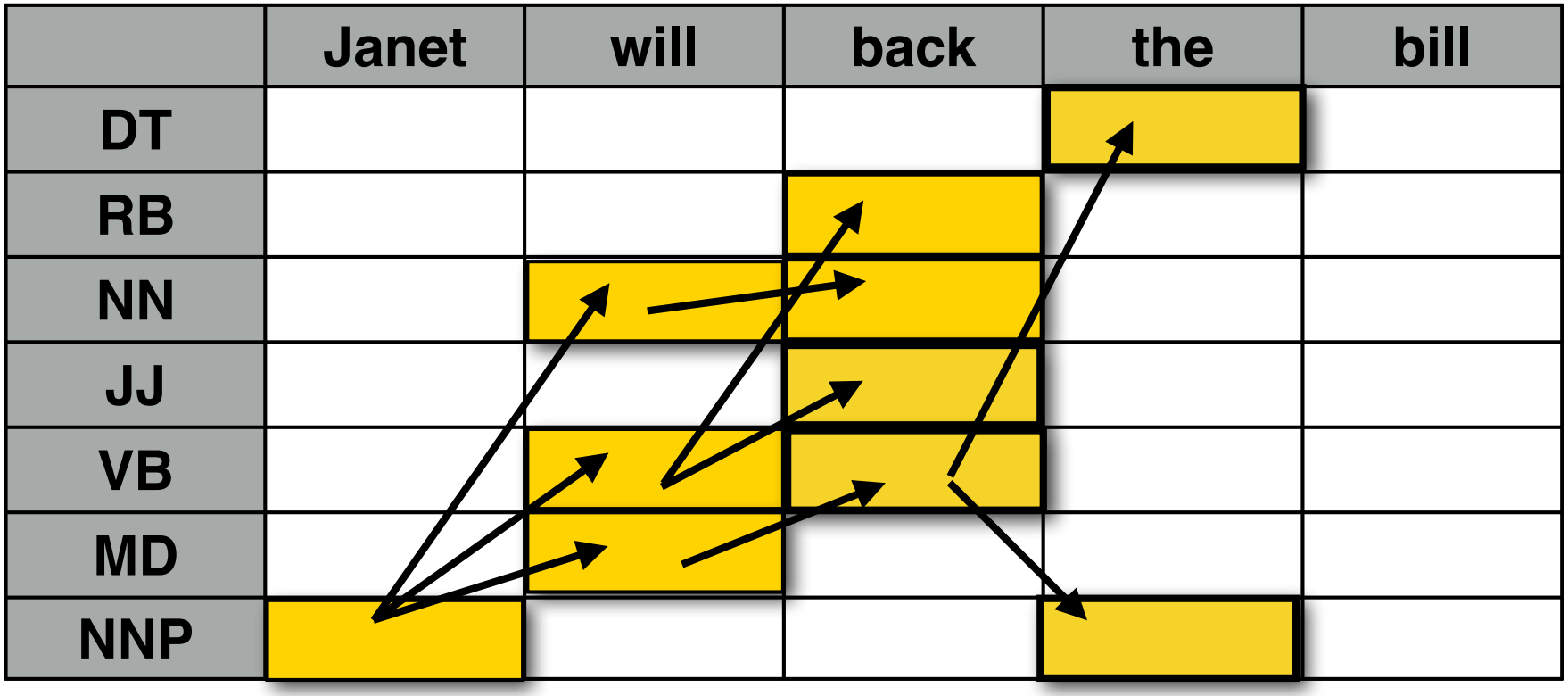

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

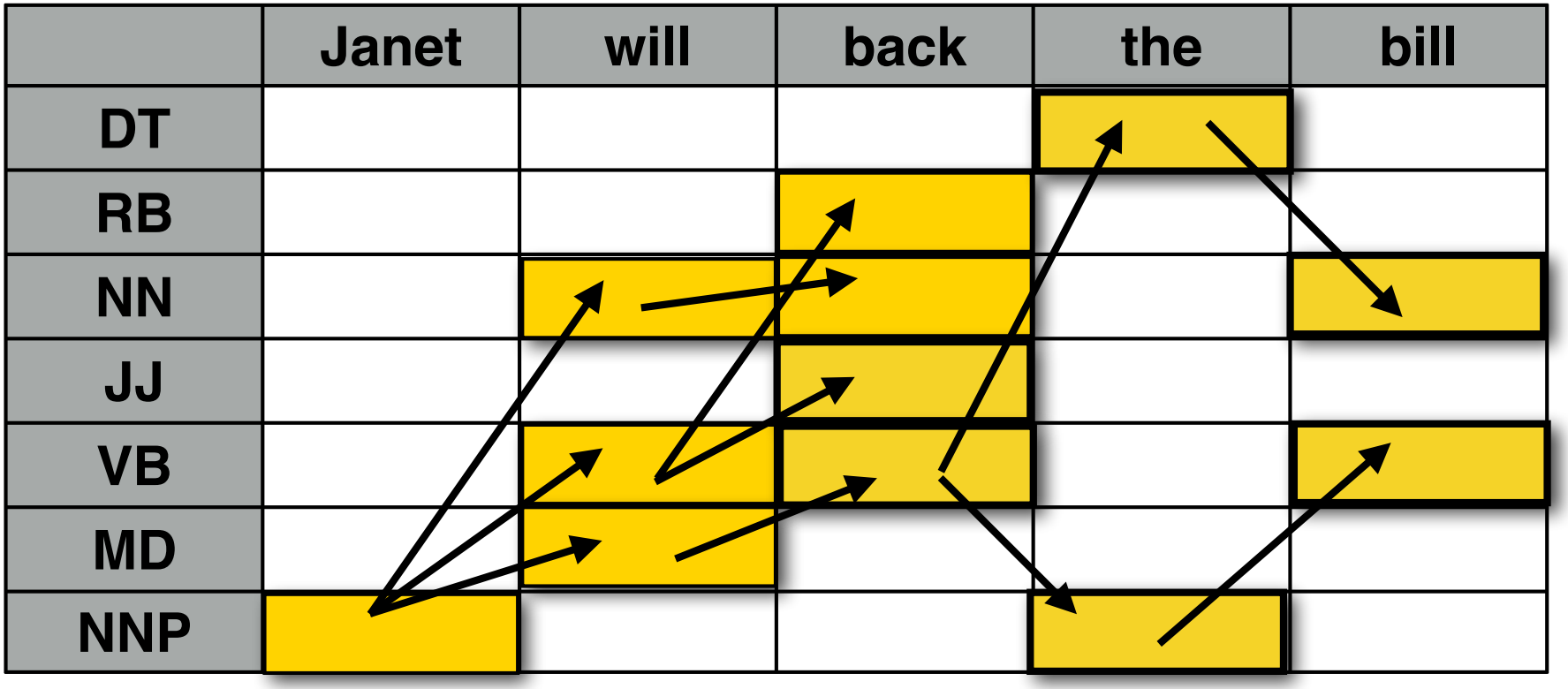

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

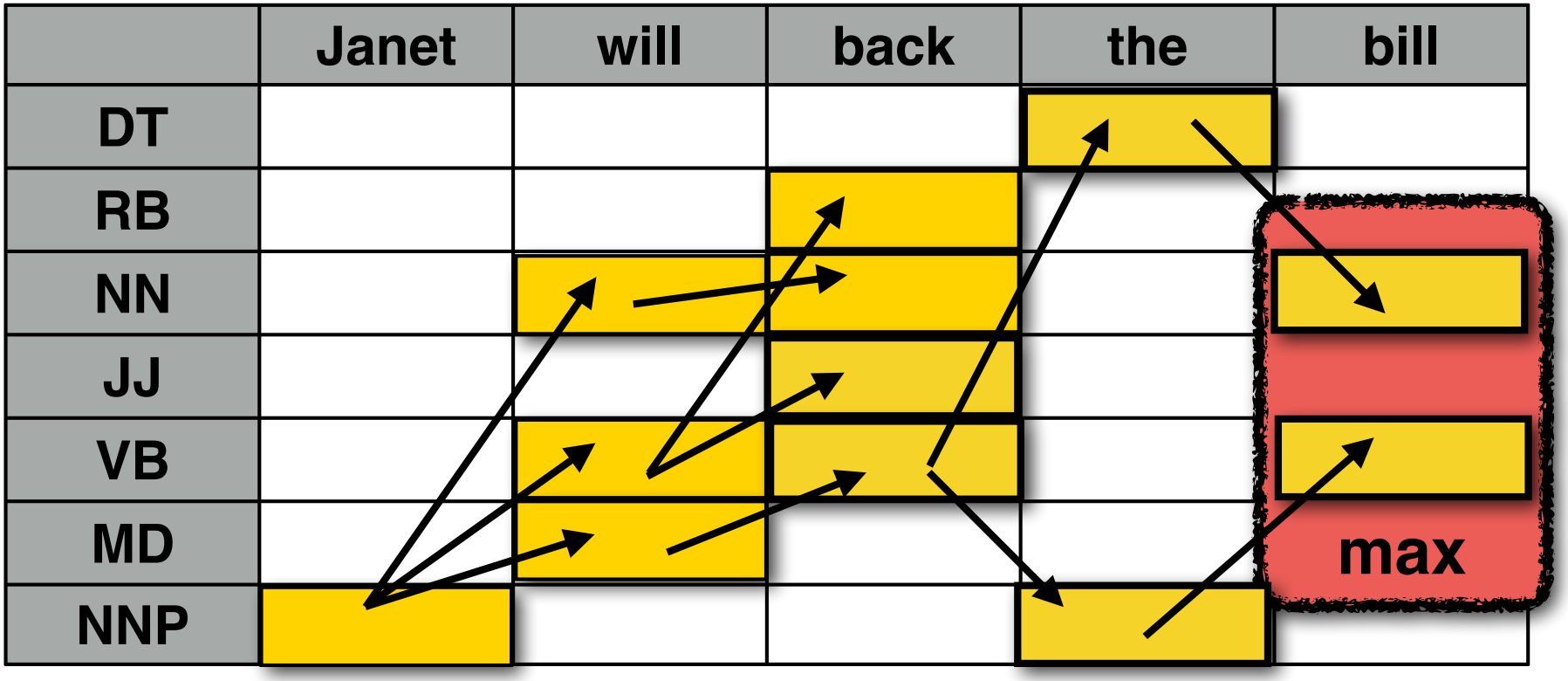

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

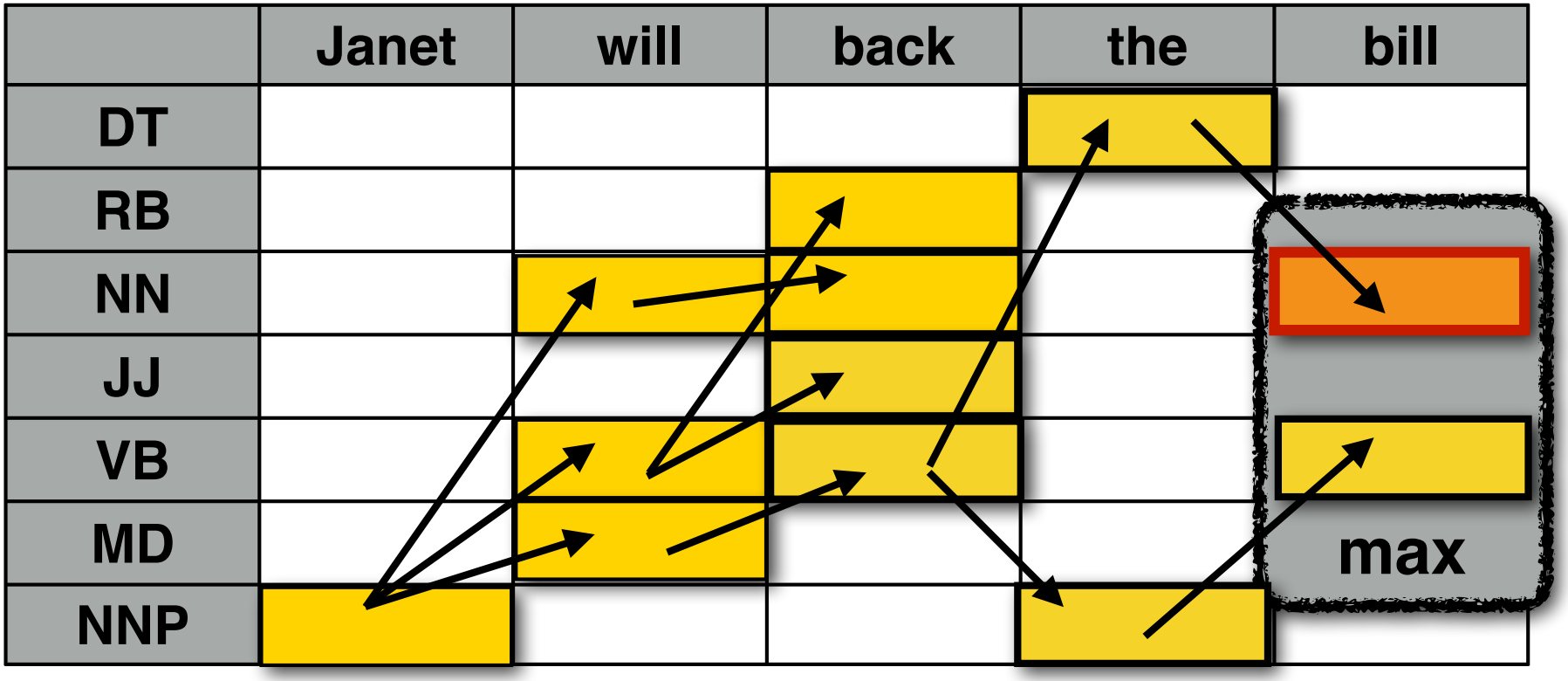

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

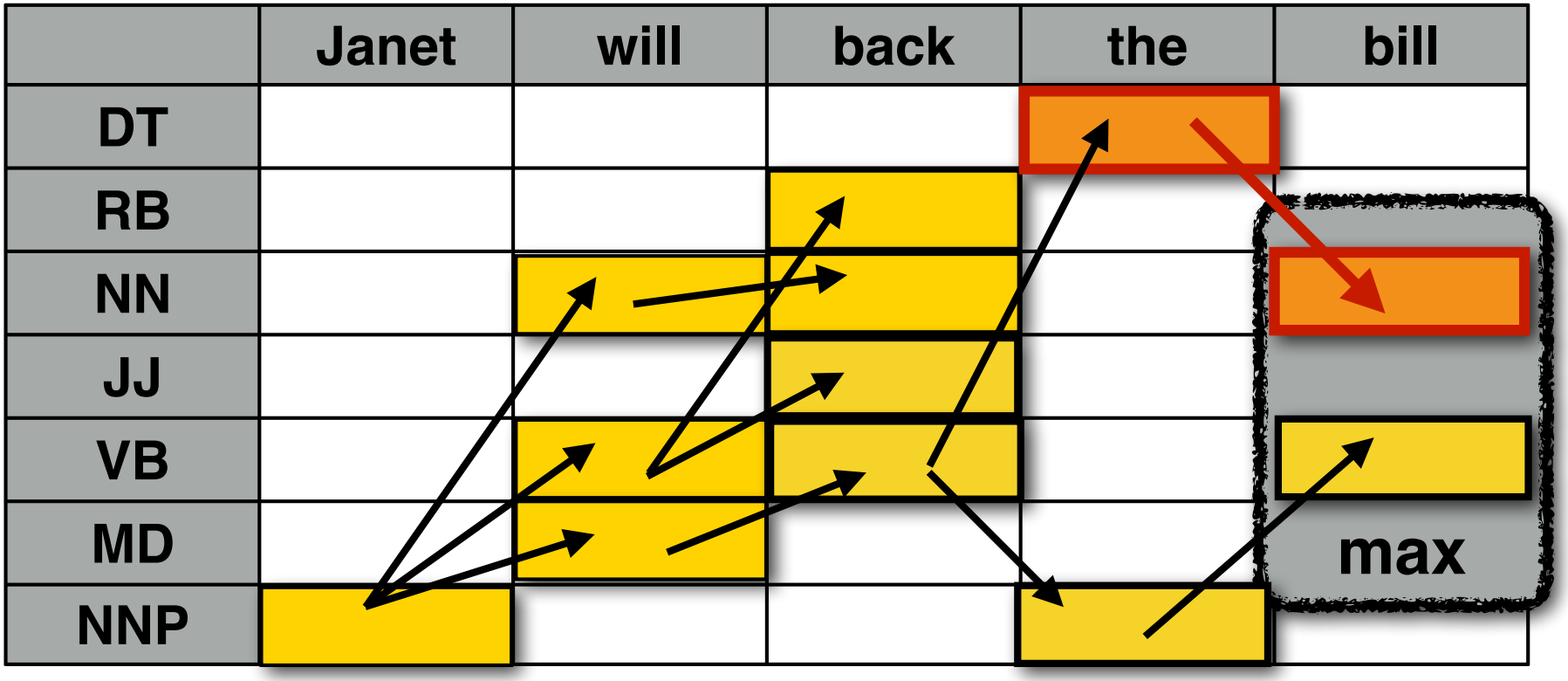

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

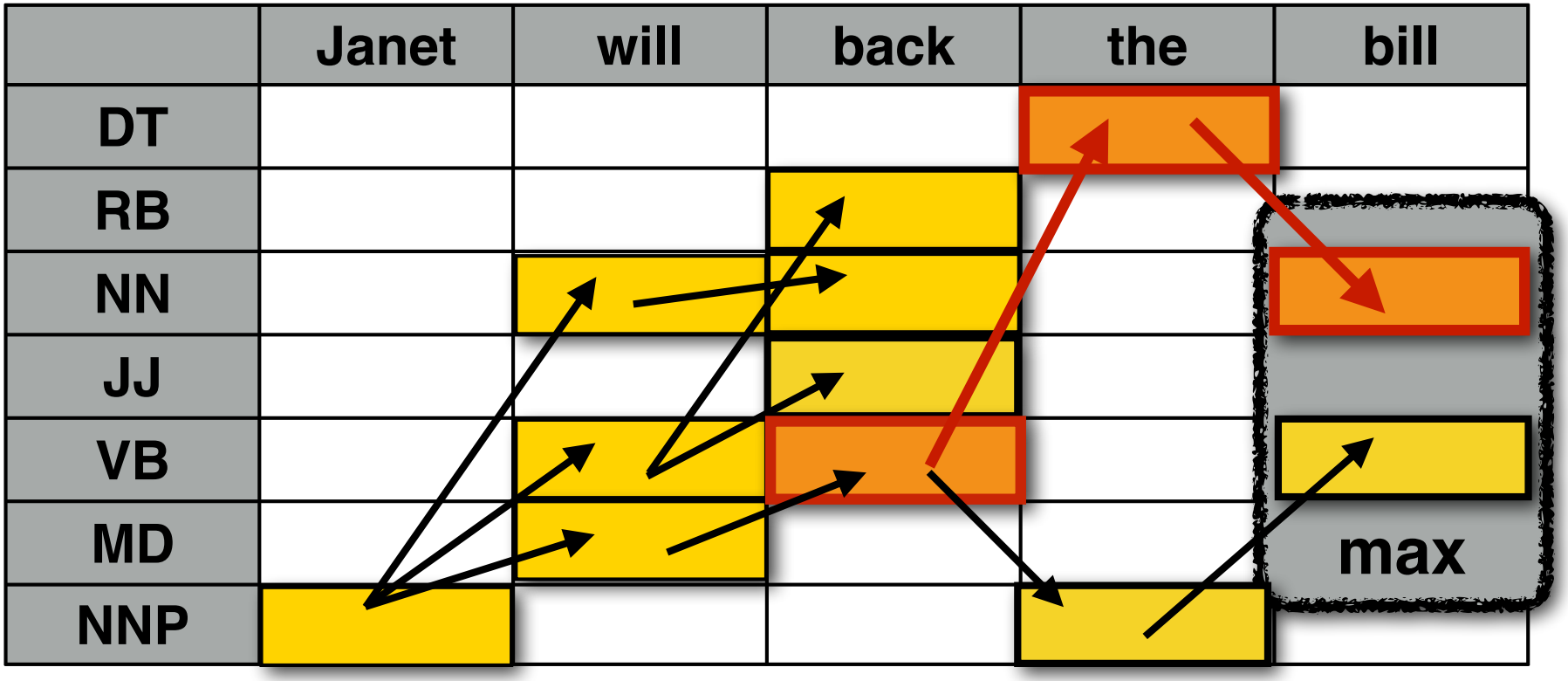

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

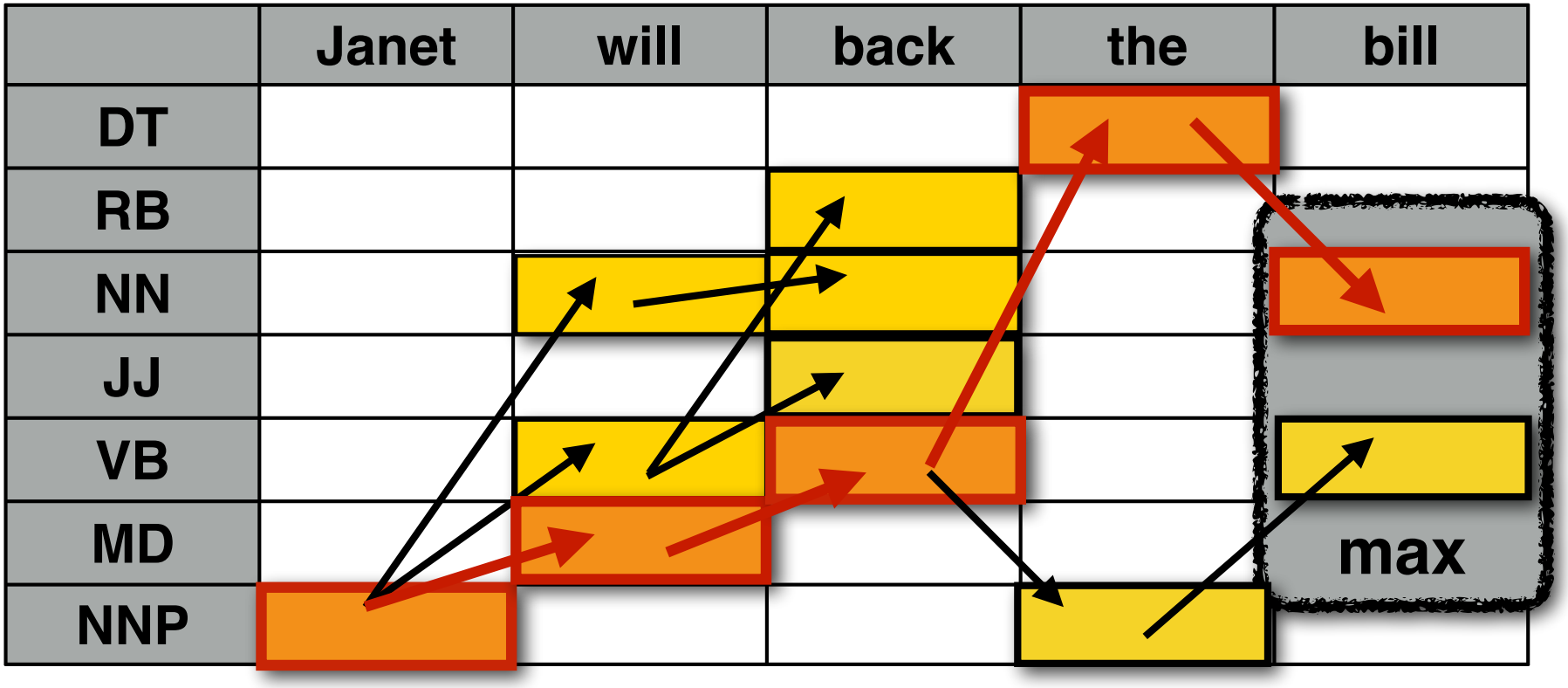

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP


Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max


Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max


Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max


Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max



Janet_NNP will_MD back_VB the_DT bill_NN

The Viterbi algorithm

Viterbi($w_{1\ldots n}$){

for t (1...T) // **INITIALIZATION: first column**

trellis[1][t].viterbi = p_init[t] × p_emit[t][w_1]

for i (2...n){ // **RECURSION: every other column**

for t (1....T){

trellis[i][t] = 0

for t' (1...T){

tmp = trellis[i-1][t'].viterbi × p_trans[t'][t]

if (tmp > trellis[i][t].viterbi){

trellis[i][t].viterbi = tmp

trellis[i][t].backpointer = t'}}

trellis[i][t].viterbi ×= p_emit[t][w_i]}}

t_max = NULL, vit_max = 0; // **FINISH: find the best cell in the last column**

for t (1...T)

if (trellis[n][t].vit > vit_max){t_max = t; vit_max = trellis[n][t].value }

return **unpack**(n, t_max);

}

Viterbi for Trigram HMMs

In a Trigram HMM, transition probabilities are of the form:

$$P(t^{(i)} = t^i \mid t^{(i-1)} = t_j, t^{(i-2)} = t_k)$$

The i-th tag in the sequence influences the probabilities of the (i+1)-th tag and the (i+2)-th tag:

$$\ldots P(t^{(i+1)} \mid \mathbf{t^{(i)}}, t^{(i-1)}) \ldots P(t^{(i+2)} \mid t^{(i+1)}, \mathbf{t^{(i)}})$$

Hence, each row in the trellis for a trigram HMM has to correspond to a pair of tags — the current and the preceding tag:

(abusing notation)

$\text{trellis}[i]\langle j,k\rangle$: word $w^{(i)}$ has tag t_j, word $w^{(i-1)}$ has tag t_k

The trellis now has T^2 rows.

But we still need to consider only T transitions into each cell, since the current word's tag is the next word's preceding tag:

Transitions are only possible from $\text{trellis}[i]\langle \mathbf{j},k\rangle$ to $\text{trellis}[i+1]\langle l,\mathbf{j}\rangle$

The three basic problems for HMMs

Given an **output sequence** $\boldsymbol{w}=w^{(1)}\ldots w^{(N)}$:

$\boldsymbol{w}$="she promised to back the bill"

Problem I (Likelihood): find $P(w \mid \lambda)$

Given an HMM $\lambda = (A, B, \pi)$, compute the likelihood of the observed output, $P(\boldsymbol{w} \mid \lambda)$

Problem II (Decoding i.e. Tagging): find $Q=q^{(1)}..q^{(N)}$

Given an HMM $\lambda = (A, B, \pi)$, what is the most likely sequence of states $Q=q^{(1)}..q^{(N)} \approx t^{(1)}...t^{(N)}$ to generate $\boldsymbol{w}$?

Problem III (MLE Estimation): find $argmax_{\lambda} P(\boldsymbol{w} \mid \lambda)$

Find the parameters A, B, π which maximize $P(\boldsymbol{w} \mid \lambda)$

Dynamic programming algorithms for HMMs

I. Likelihood of the input:

Compute $P(\boldsymbol{w}|\ \lambda\ )$ for an $\boldsymbol{w}$ and HMM λ

⇒ **Forward algorithm**

II. Decoding (=tagging) the input:

Find best tags $\boldsymbol{t}^* = argmax_{\boldsymbol{t}}\ P(\boldsymbol{t}\ |\ \boldsymbol{w},\lambda)$ for input $\boldsymbol{w}$ and HMM λ

⇒ **Viterbi algorithm**

III. Estimation (=learning the model):

Find best model parameters $\lambda^* = argmax_{\lambda}\ P(\boldsymbol{t},\ \boldsymbol{w}\ |\ \lambda)$
for unlabeled training data $\boldsymbol{w}$

⇒ **Forward-Backward algorithm**

Computing $P(\mathbf{w})$: the Forward algorithm

To compute the probability of a sentence according to an HMM, we have to sum over all possible tag sequences: $P(\mathbf{w}) = \sum_{\mathbf{t}} P(\mathbf{w}, \mathbf{t})$

The **Forward algorithm** computes this sum efficiently:

Base case: For the **first word** in the sentence, and for each tag j:

$$\text{forward}[1][j] = \pi(t_j) P(w^{(1)} \mid t_j)$$

Recurrence: For **any other word** i, and for each tag j:

$$\text{forward}[i][j] = P(w^{(i)} \mid t_j) \sum_k \text{forward}[i-1][k] P(t_j \mid t_k)$$

End: For the **last word** in the sentence, and for all tags k:

$$P(\mathbf{w}) = \sum_k \text{forward}[N][k]$$

Same as Viterbi, except sum instead of max