Language and Statistics II

Lecture 13: Deductive Parsing, Especially with Weights

Noah Smith

Remember Prolog?

Invented around 1972 for AI and CL.

Horn clauses:

```
father(X,Y) :- parents(X,_,Y),male(X).
grandfather(X,Z) :-
  father(X,Y),father(Y,Z).
male(john).
male(joe).
parents(william,jane,john).
parents(joe,mary,william).
```

Prolog

- Given a query, Prolog tries to prove some instantiation of it.

```
?- grandpa(X,john).
  X=joe;
  yes.
```

- Prolog uses a particular brand of **search**, which we won't talk about.
- The point: deduction and theorem proving are a useful way to **describe** algorithms in CL.

Parsing as Deduction (high-level view)

- *Axioms*: the sentence s, the grammar G
- *Inference rules*: correspond to the parsing algorithm
- *Theorems*: partial parses, then stored in **chart**
- *Stopping criterion*: “goal” reached or agenda empty
- *Output*: true iff $s \in L(G)$ (i.e., “goal” is proven), along with a proof

Shieber, Schabes, & Pereira (1995): use Prolog!

- For efficiency, minimize redundancy in chart and agenda.

Example 1: top-down parsing

Item form: $[\bullet \beta, j]$

Axioms: $[\bullet S, 0]$

Goals: $[\bullet \,, n]$

Inference rules:

Scanning $$\frac{[\bullet\, w_{j+1}\beta, j]}{[\bullet\, \beta, j+1]}$$

Prediction $$\frac{[\bullet\, B\beta, j]}{[\bullet\, \gamma\beta, j]} \quad B \to \gamma$$

[•β, j] means: "Starting after word j, I need to build β (a sequence of symbols)."

Scanning: match the next word in the sentence to the next required symbol.

Prediction: use grammar to see how next required symbol could expand.

What would happen if we implemented this?

Example 2: shift-reduce parsing

Item form: $[\alpha \bullet, j]$

Axioms: $[\bullet, 0]$

Goals: $[S \bullet, n]$

Inference Rules:

Shift $\dfrac{[\alpha \bullet, j]}{[\alpha w_{j+1} \bullet, j+1]}$

Reduce $\dfrac{[\alpha\gamma \bullet, j]}{[\alpha B \bullet, j]} \quad B \to \gamma$

[α•, j] means: "Up to word j, I have begun to build α (a stack of symbols)."

Shift (scanning): move the next word in the sentence to the stack.

Reduce (completion): use grammar to see how the top symbols on the stack can be combined.

Algorithm	*Bottom-Up*	*Top-Down*	*Earley's*
Item form	$[\alpha \bullet, j]$	$[\bullet \beta, j]$	$[i, A \rightarrow \alpha \bullet \beta, j]$
Invariant	$\alpha w_{j+1} \cdots w_n \overset{*}{\Rightarrow} w_1 \cdots w_n$	$S \overset{*}{\Rightarrow} w_1 \cdots w_j \beta$	$S \overset{*}{\Rightarrow} w_1 \cdots w_i A \gamma$ $\alpha w_{j+1} \cdots w_n \overset{*}{\Rightarrow} w_{i+1} \cdots w_n$
Axioms	$[\bullet, 0]$	$[\bullet S, 0]$	$[0, S' \rightarrow \bullet S, 0]$
Goals	$[S \bullet, n]$	$[\bullet, n]$	$[0, S' \rightarrow S \bullet, n]$
Scanning	$\frac{[\alpha \bullet, j]}{[\alpha w_{j+1} \bullet, j+1]}$	$\frac{[\bullet w_{j+1} \beta, j]}{[\bullet \beta, j+1]}$	$\frac{[i, A \rightarrow \alpha \bullet w_{j+1} \beta, j]}{[i, A \rightarrow \alpha w_{j+1} \bullet \beta, j+1]}$
Prediction		$\frac{[\bullet B \beta, j]}{[\bullet \gamma \beta, j]} \quad B \rightarrow \gamma$	$\frac{[i, A \rightarrow \alpha \bullet B \beta, j]}{[j, B \rightarrow \bullet \gamma, j]} \quad B \rightarrow \gamma$
Completion	$\frac{[\alpha \gamma \bullet, j]}{[\alpha B \bullet, j]} \quad B \rightarrow \gamma$		$\frac{[i, A \rightarrow \alpha \bullet B \beta, k] \quad [k, B \rightarrow \gamma \bullet, j]}{[i, A \rightarrow \alpha B \bullet \beta, j]}$

From Shieber, Schabes, & Pereira (1995)

Earley's Algorithm, etc.

- A more efficient combination of two naïve algorithms. (A more efficient logic program.)

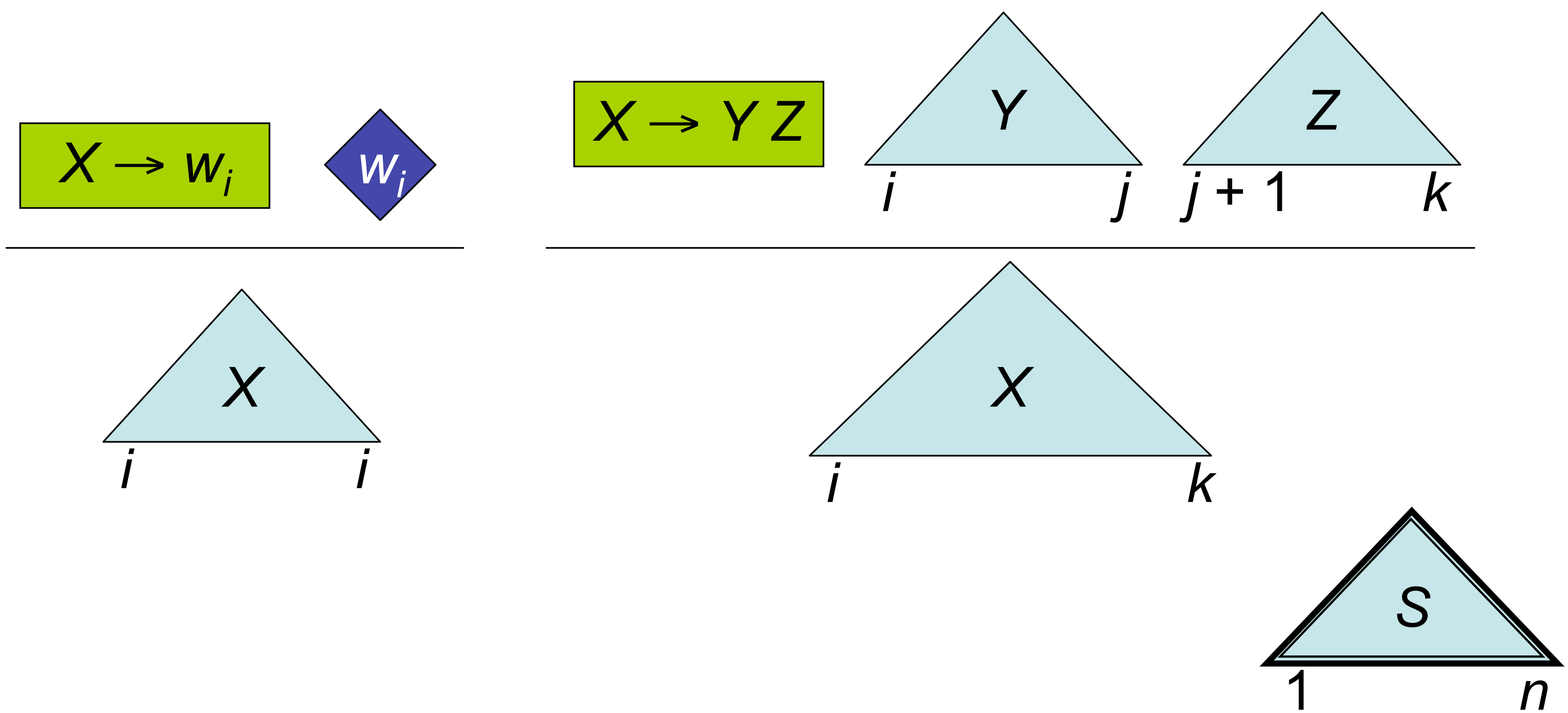
- SS&P went on to give logic programs for CCG parsing and TAG parsing.

- McAllester (2002) shows how to automatically derive asymptotic runtime and space bounds for such programs.

The Point

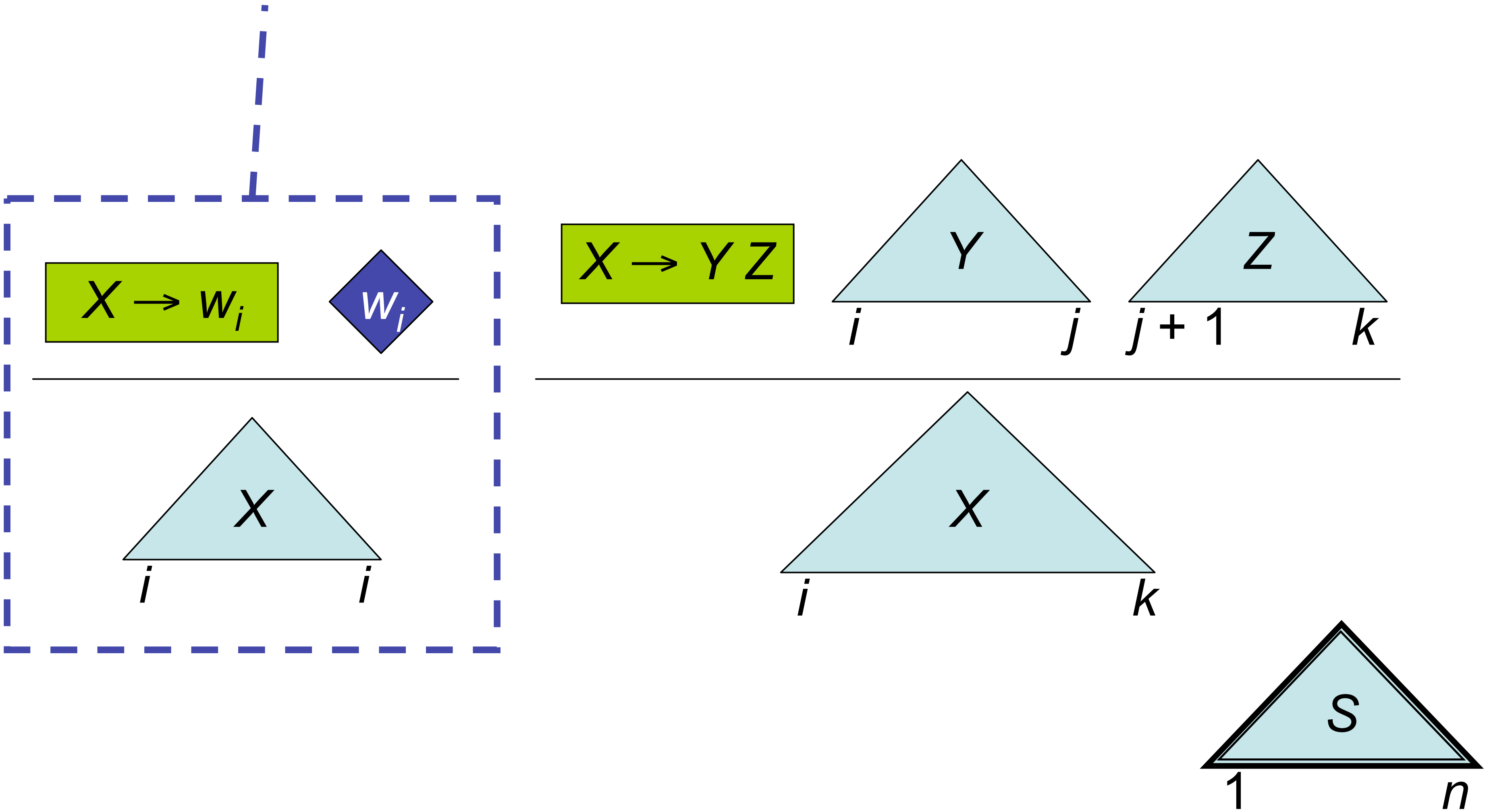
- Logic programs are helpful for
  - Algorithm specification
  - Prototype implementation
  - Finding **more efficient** algorithms

CKY



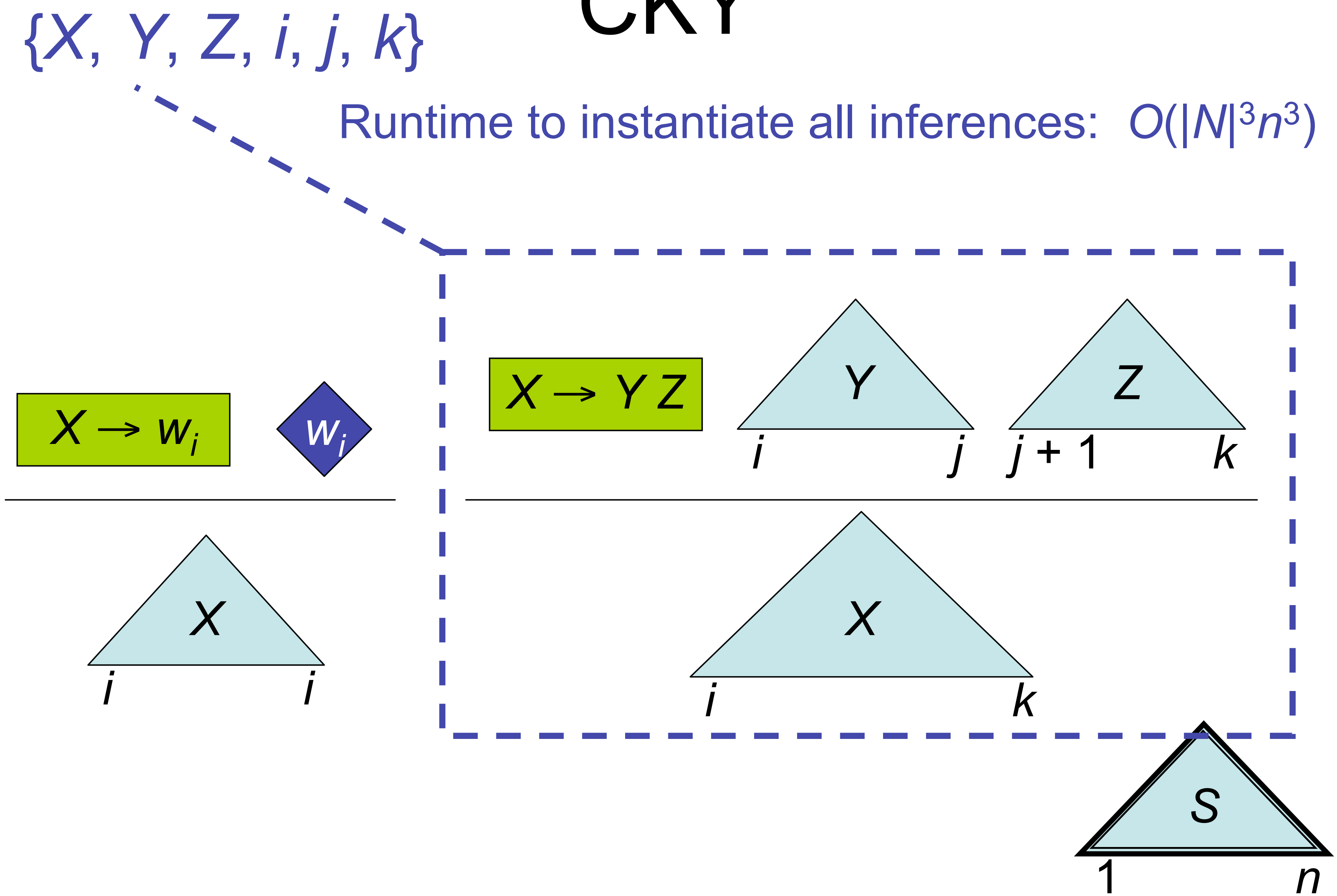
CKY

$\{X, i\}$ Runtime to instantiate all inferences: $O(|N|n)$

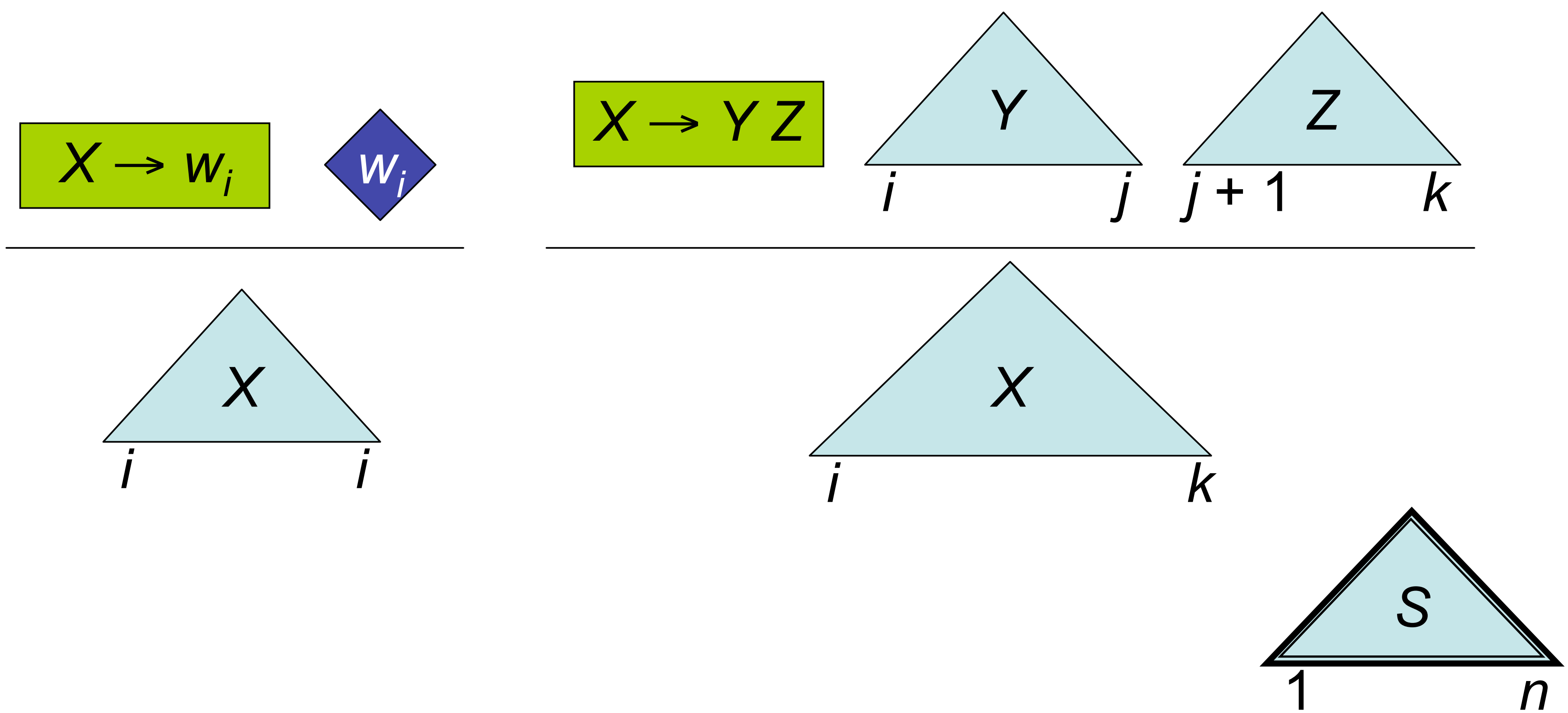


CKY

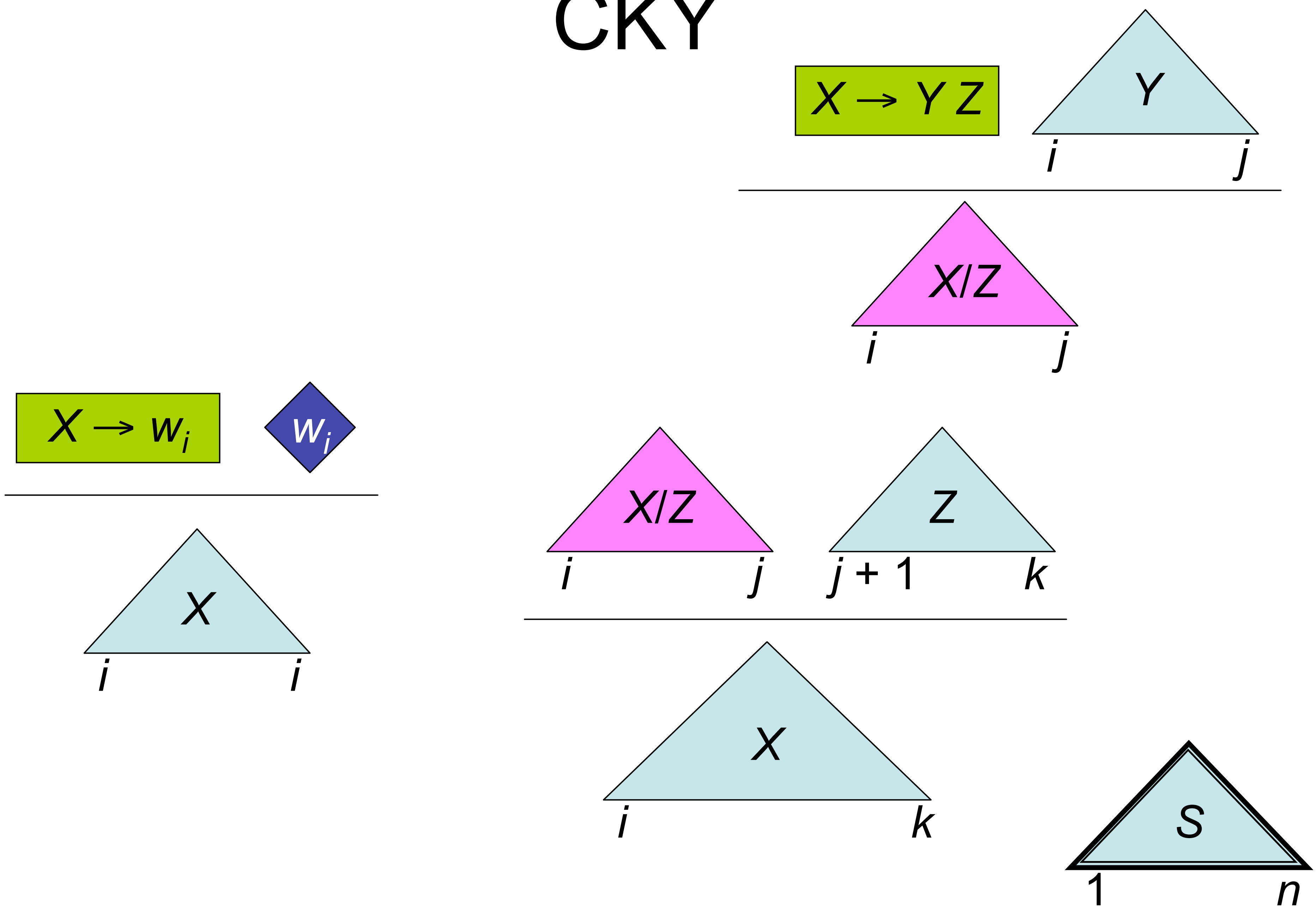
Runtime to instantiate all inferences: $O(|N|^3 n^3)$



CKY

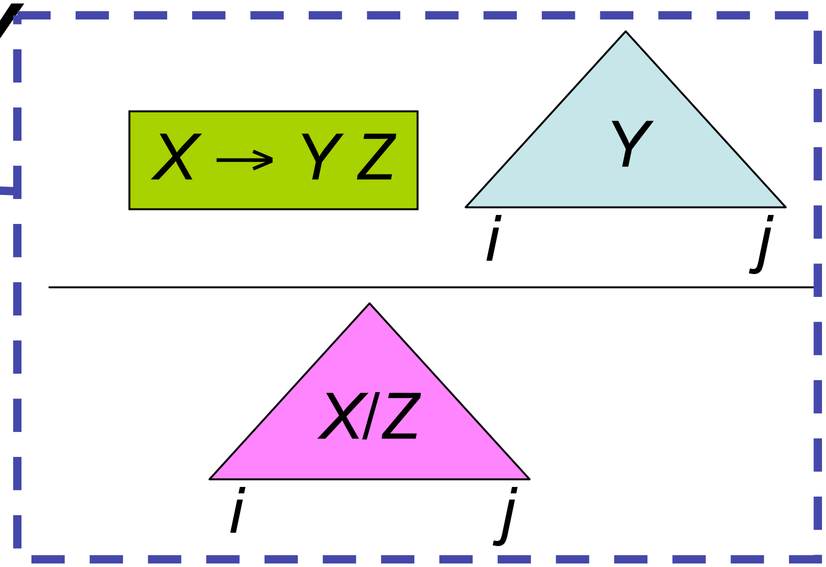


CKY

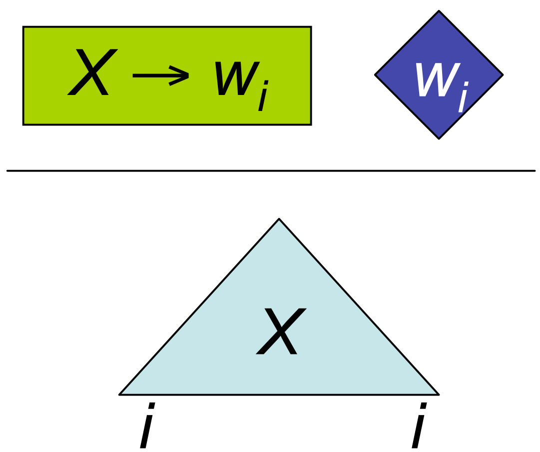


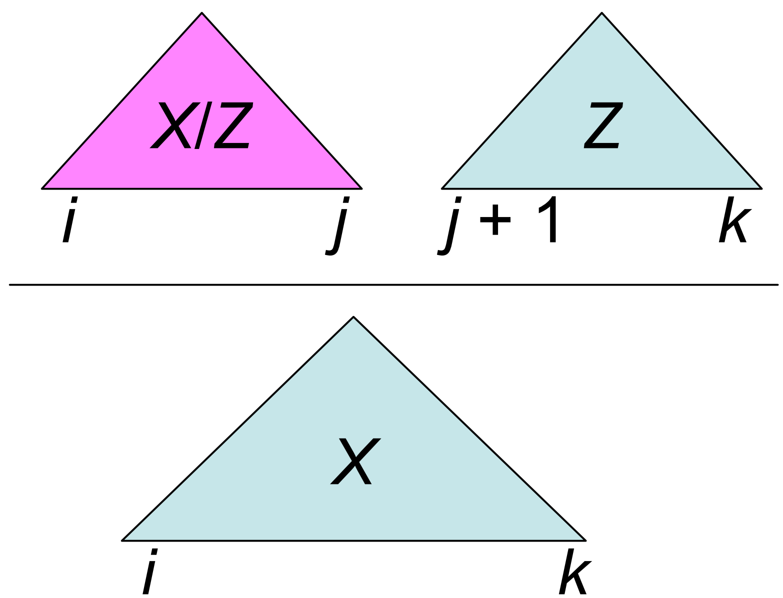
CKY

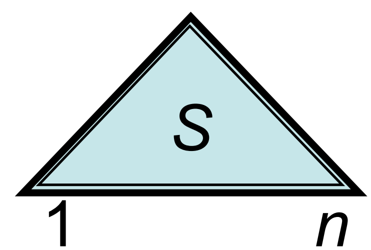
$\{X, Y, Z, i, j\}$



Runtime to instantiate all inferences: $O(|N|^3 n^2)$



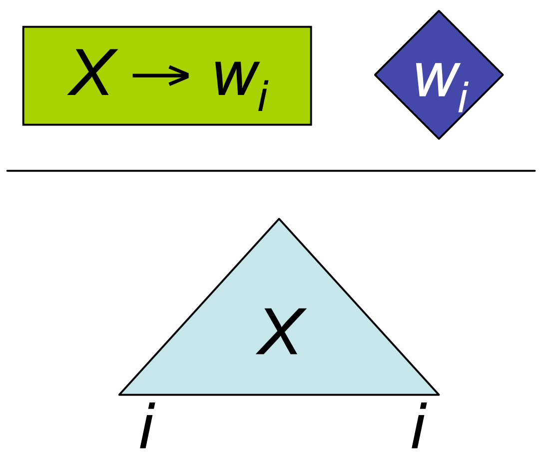


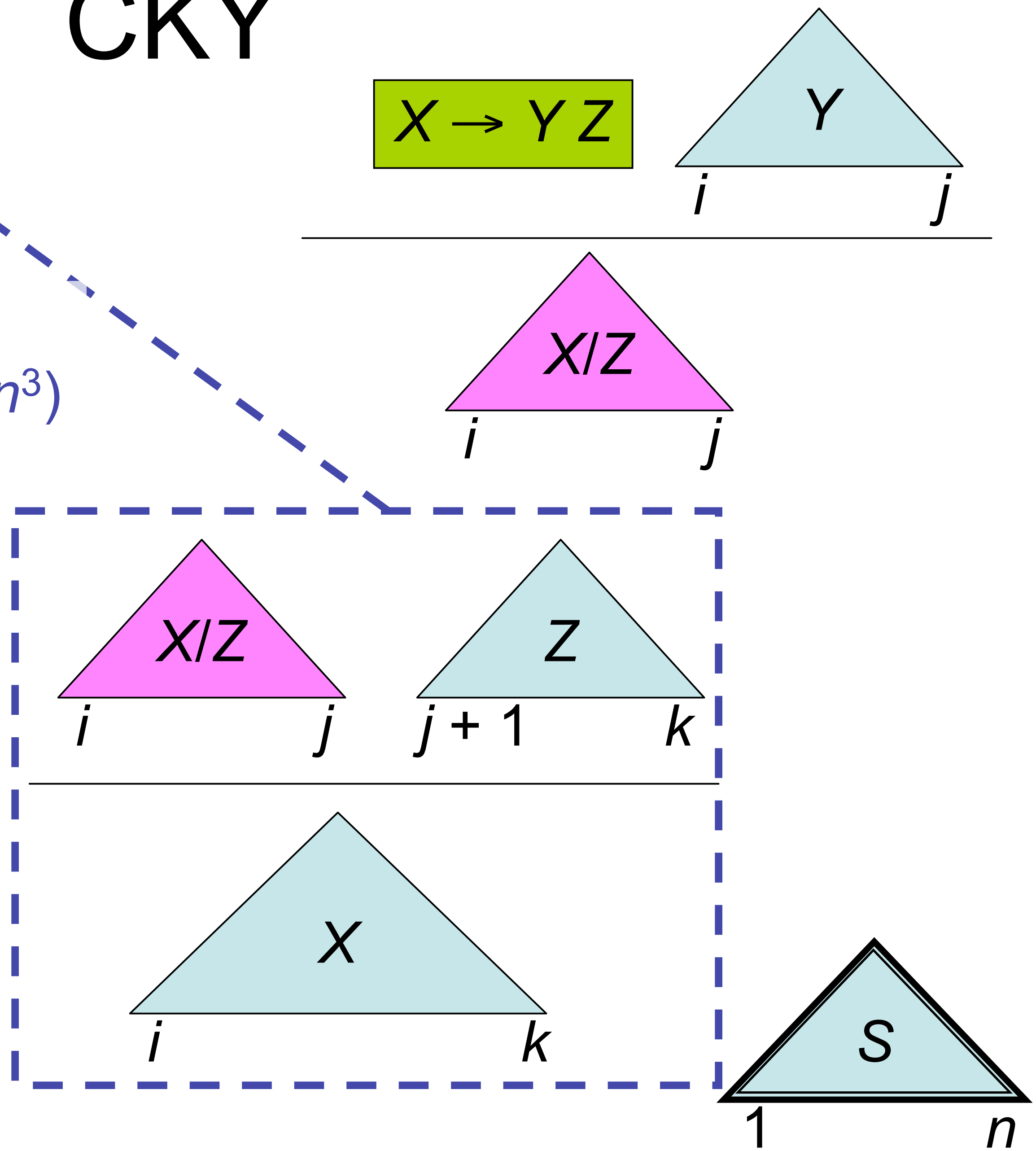


CKY

$\{X, Y, i, j, k\}$

Runtime to instantiate all inferences: $O(|N|^2 n^3)$

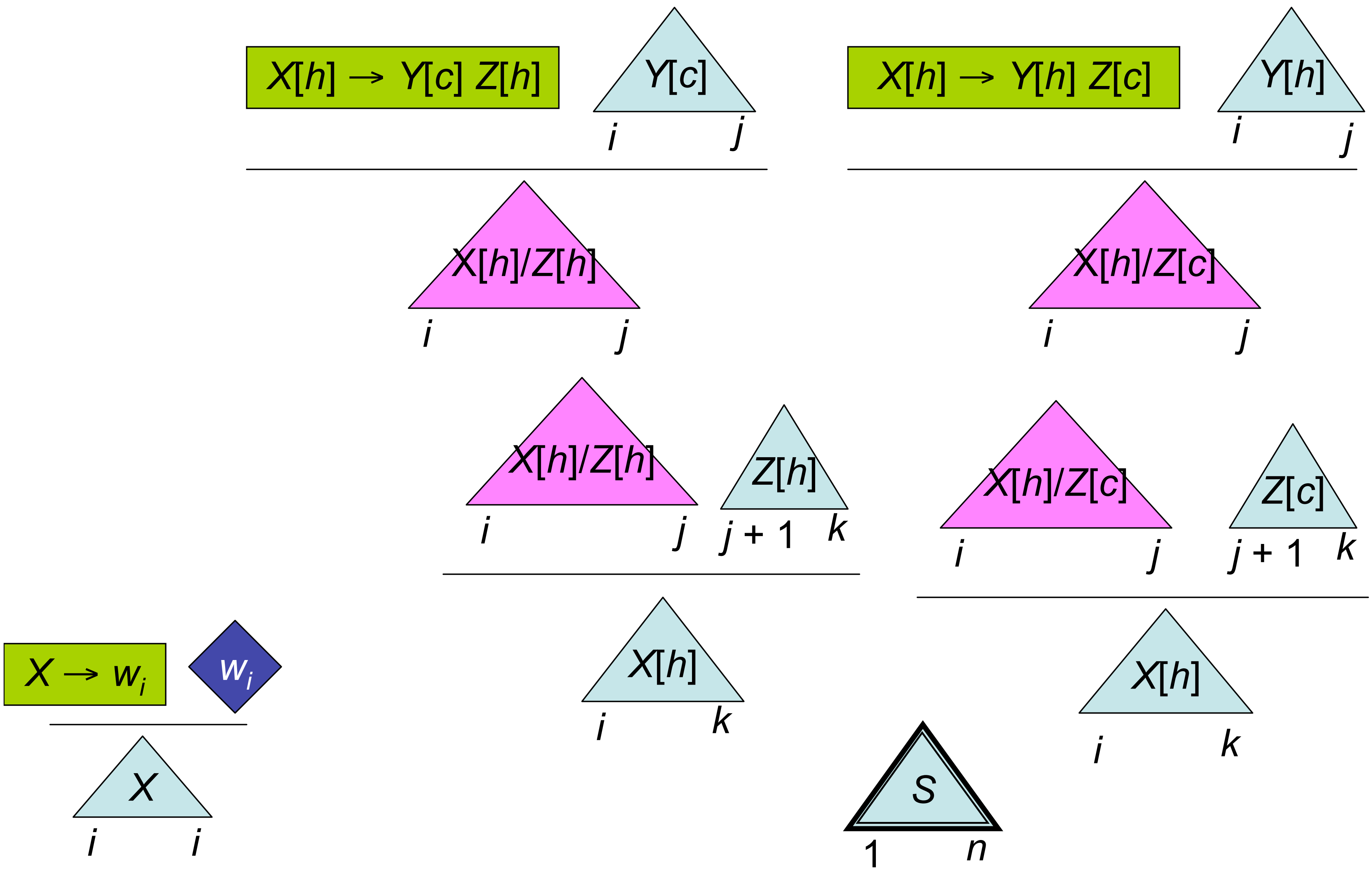




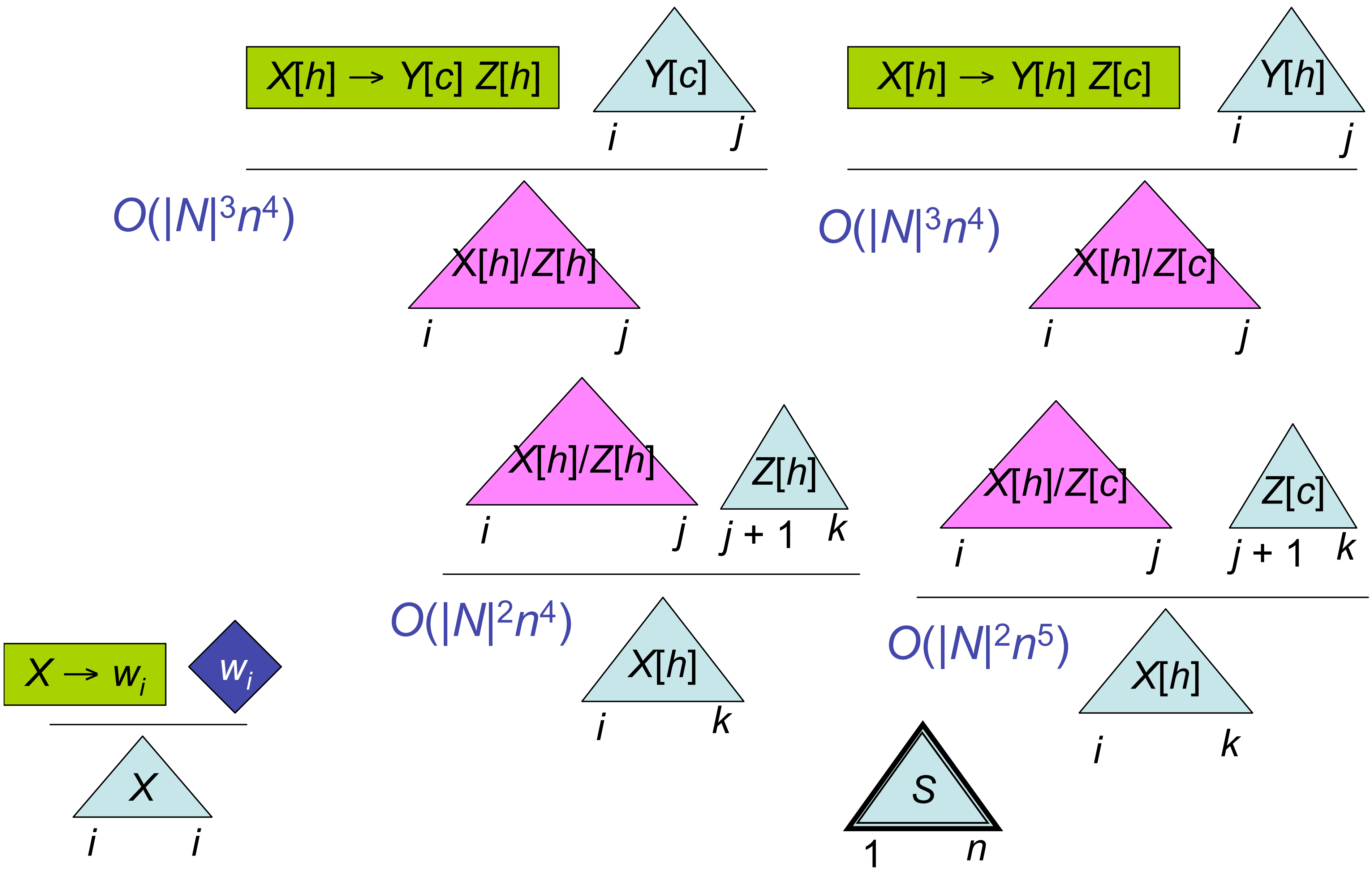
CKY

- Runtime reduced
  from $O(|N|^3 n^3)$
  to $O(|N|^2 n^3 + |N|^3 n^2)$.
- Related example: lexicalized PCFG parsing (Eisner and Satta, 1999).

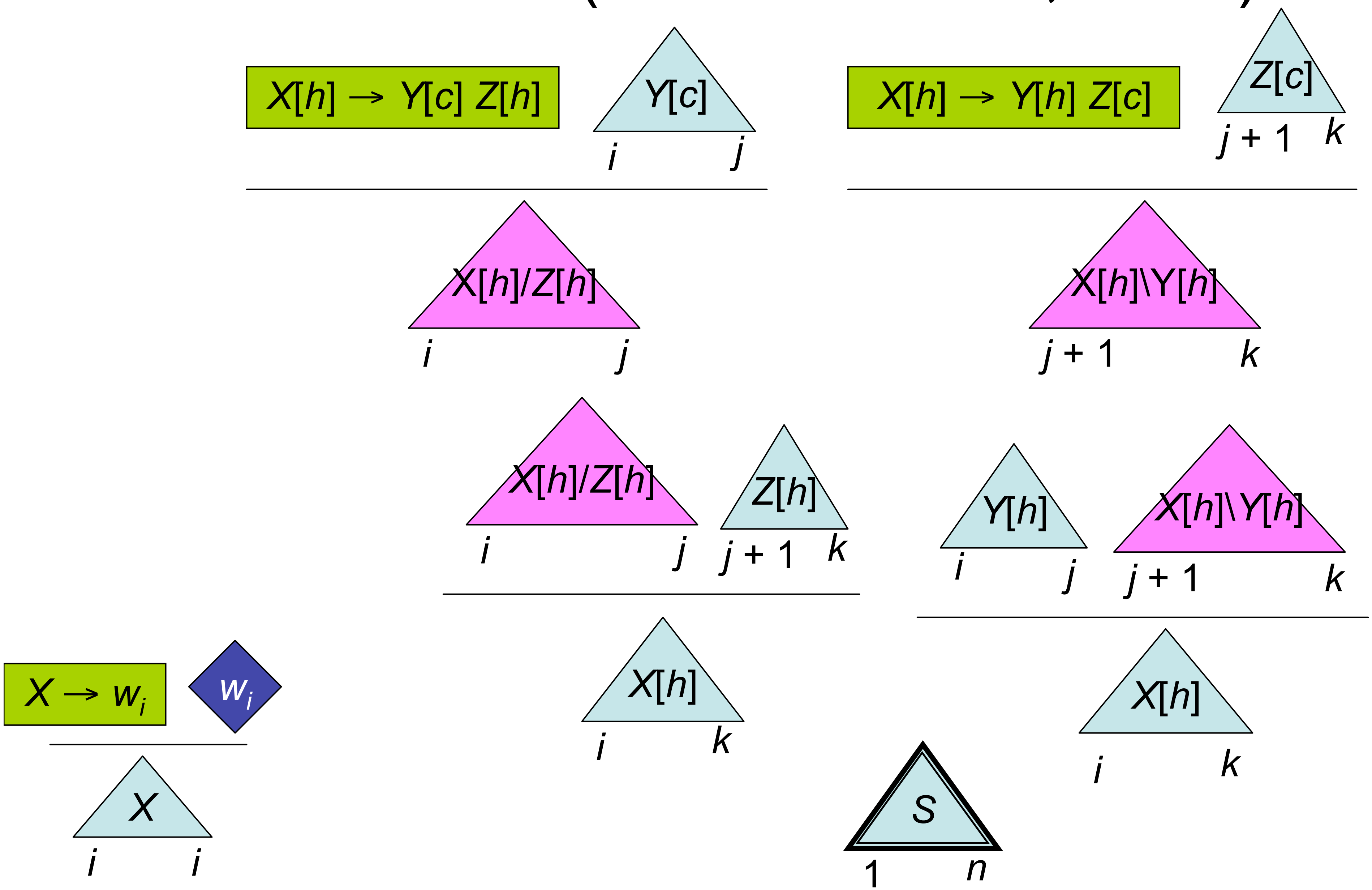
Lexicalized CKY



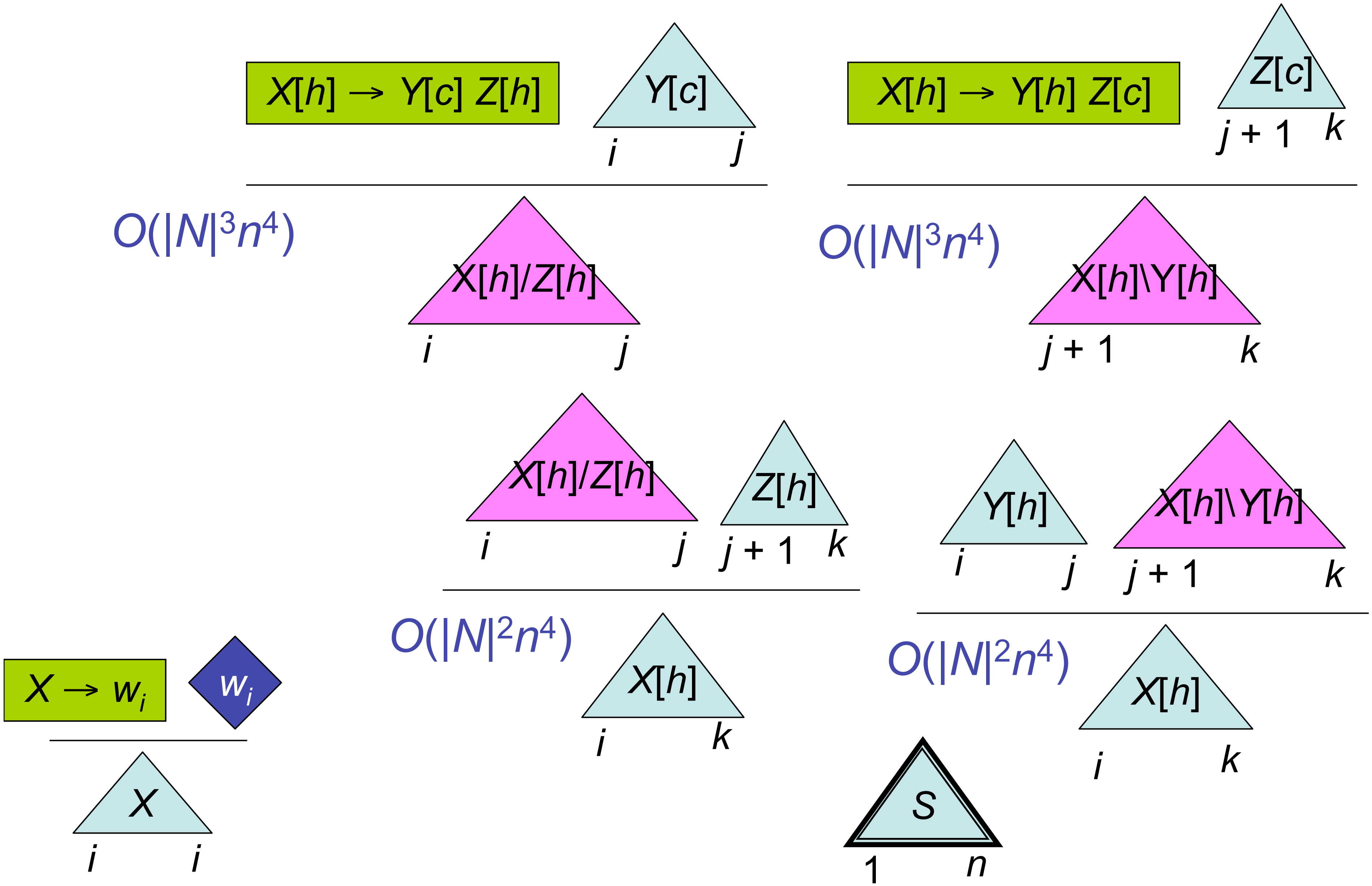
Lexicalized CKY



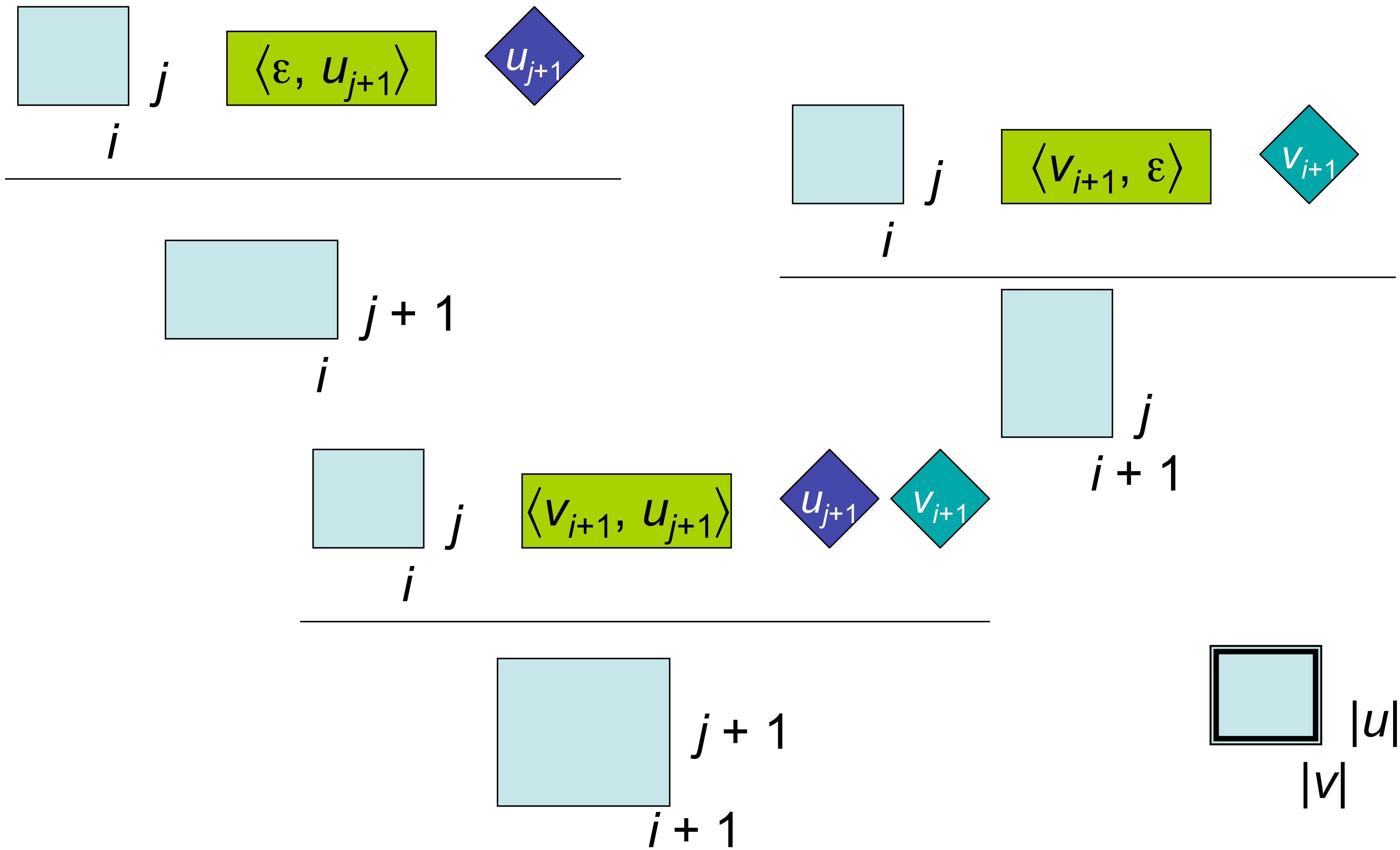
Lexicalized CKY (Eisner & Satta, 1999)



Lexicalized CKY (Eisner & Satta, 1999)

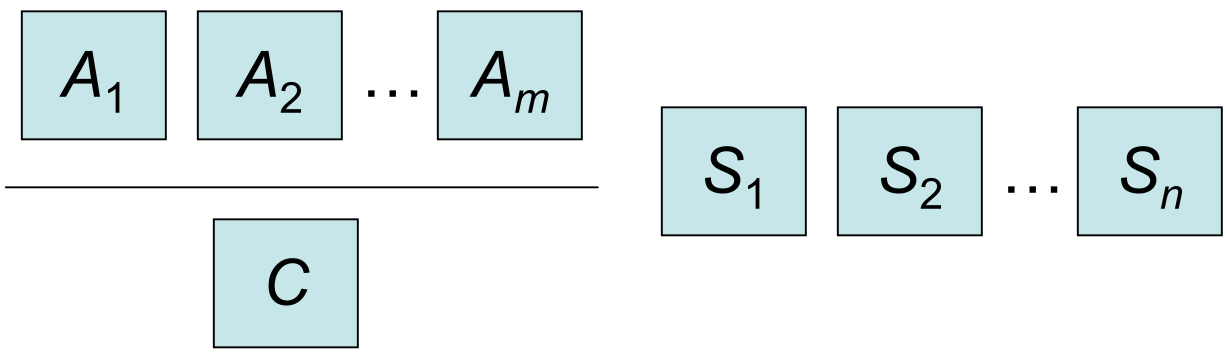


String Alignment



What about weights?

- Goodman (1999): add weights.
- Not limited to just probabilities!
  - Semiring weights.



$$V(C) \oplus= \begin{cases} \bigotimes_{i=1}^{m} V(A_i) & \text{if } \bigwedge_{j=1}^{n} \left[V(S_j) \neq \mathbf{0} \right] \\ \mathbf{0} & \text{otherwise} \end{cases}$$

Semirings

			boolean	Viterbi
set	$\mathbb{V}$	-	{false, true}	$\mathbb{R}_+$
plus	$\oplus$	associative and commutative	$\vee$	max
times	$\otimes$	associative, distributes over $\oplus$	$\wedge$	$\times$
zero	**0**	$\oplus$-identity: $x \oplus \mathbf{0} = x$	false	0
one	**1**	$\otimes$-identity: $x \otimes \mathbf{1} = x$	true	1

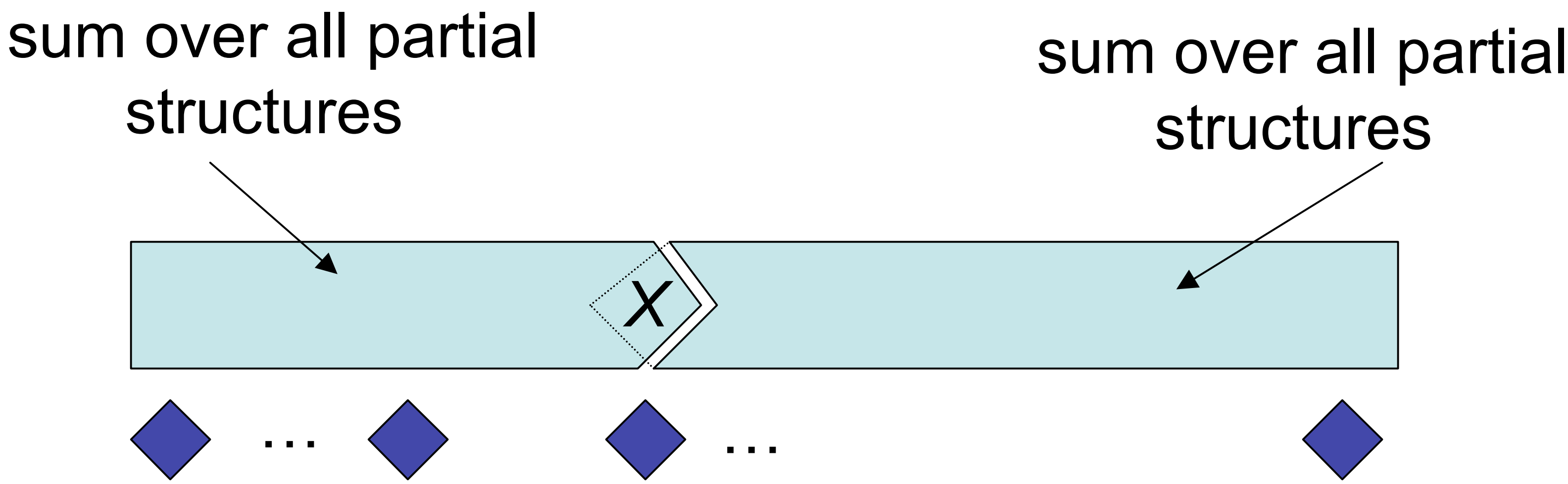
Semirings

	boolean	Viterbi	inside	counting	derivation forest
set	{false, true}	$\mathbb{R}_+$	$\mathbb{R}_+$	$\mathbb{N}_+$	$2^{\mathbb{E}}$
plus	$\vee$	max	+	+	$\cup$
times	$\wedge$	$\times$	$\times$	$\times$	concatenation
zero	false	0	0	0	$\varnothing$
one	true	1	1	1	$\{\langle\rangle\}$

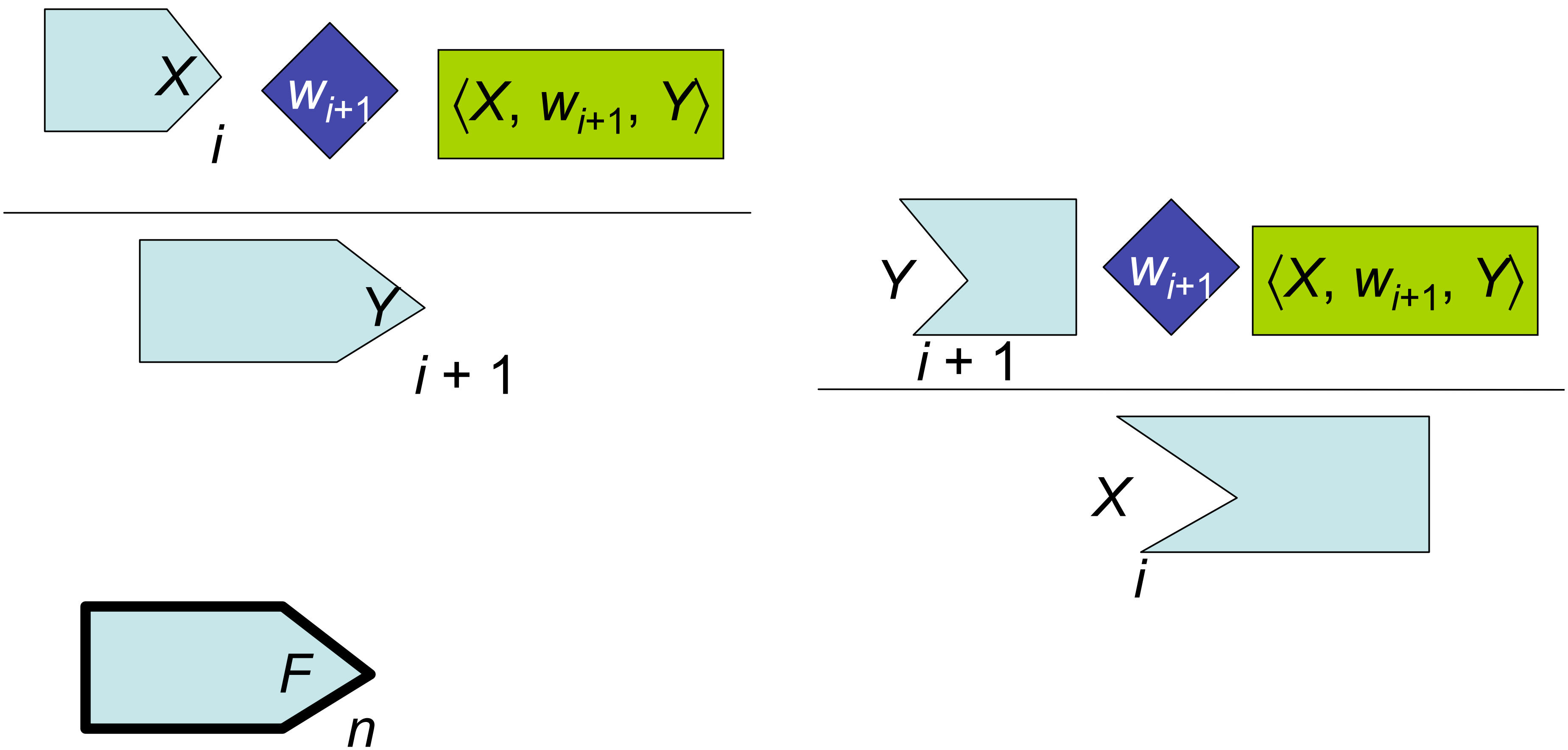
Goodman (1999)

- Viterbi-derivation and Viterbi-*n*-best semirings defined, as well.
- **Reverse** values can be computed in the same framework!

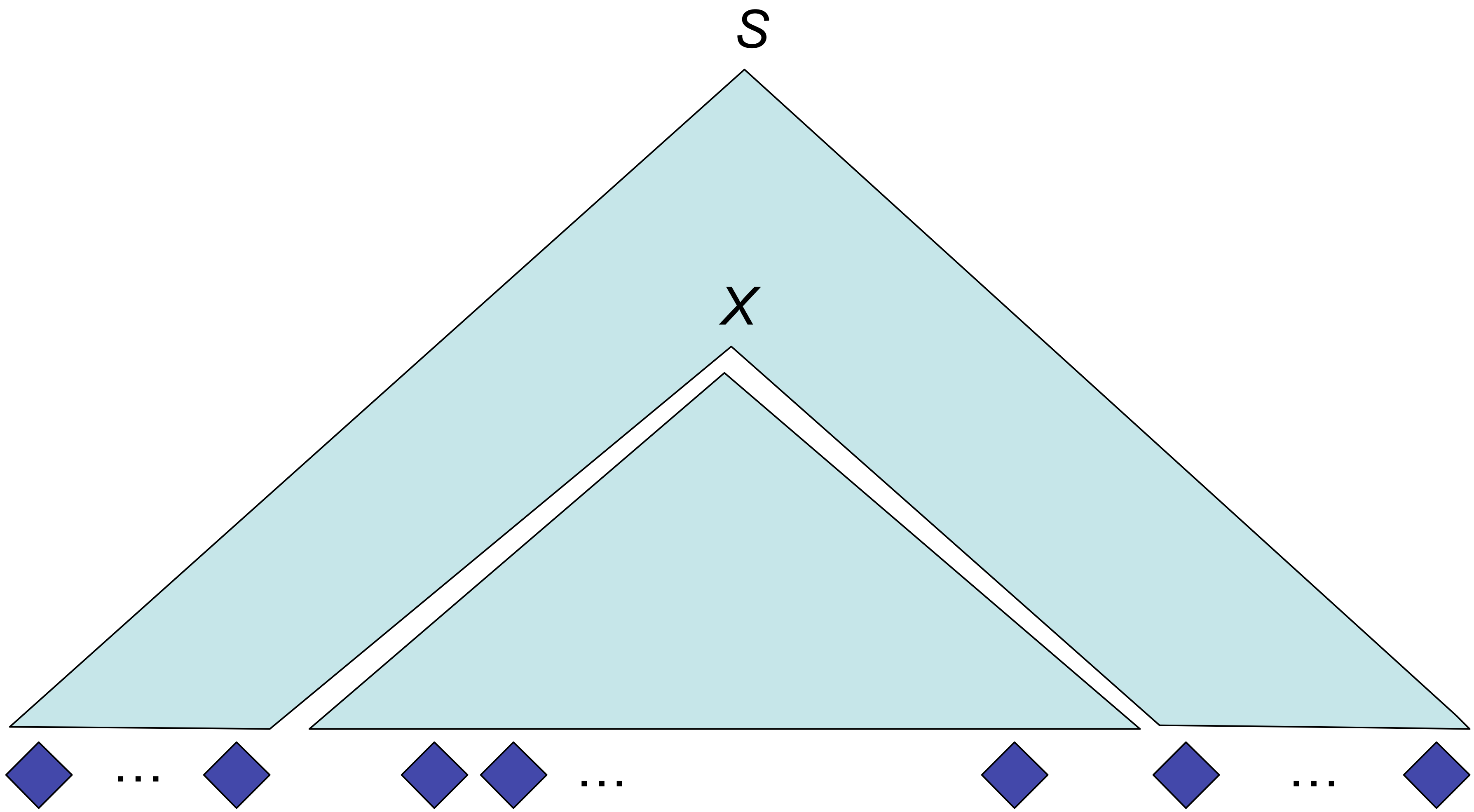
Forward and Backward Weights



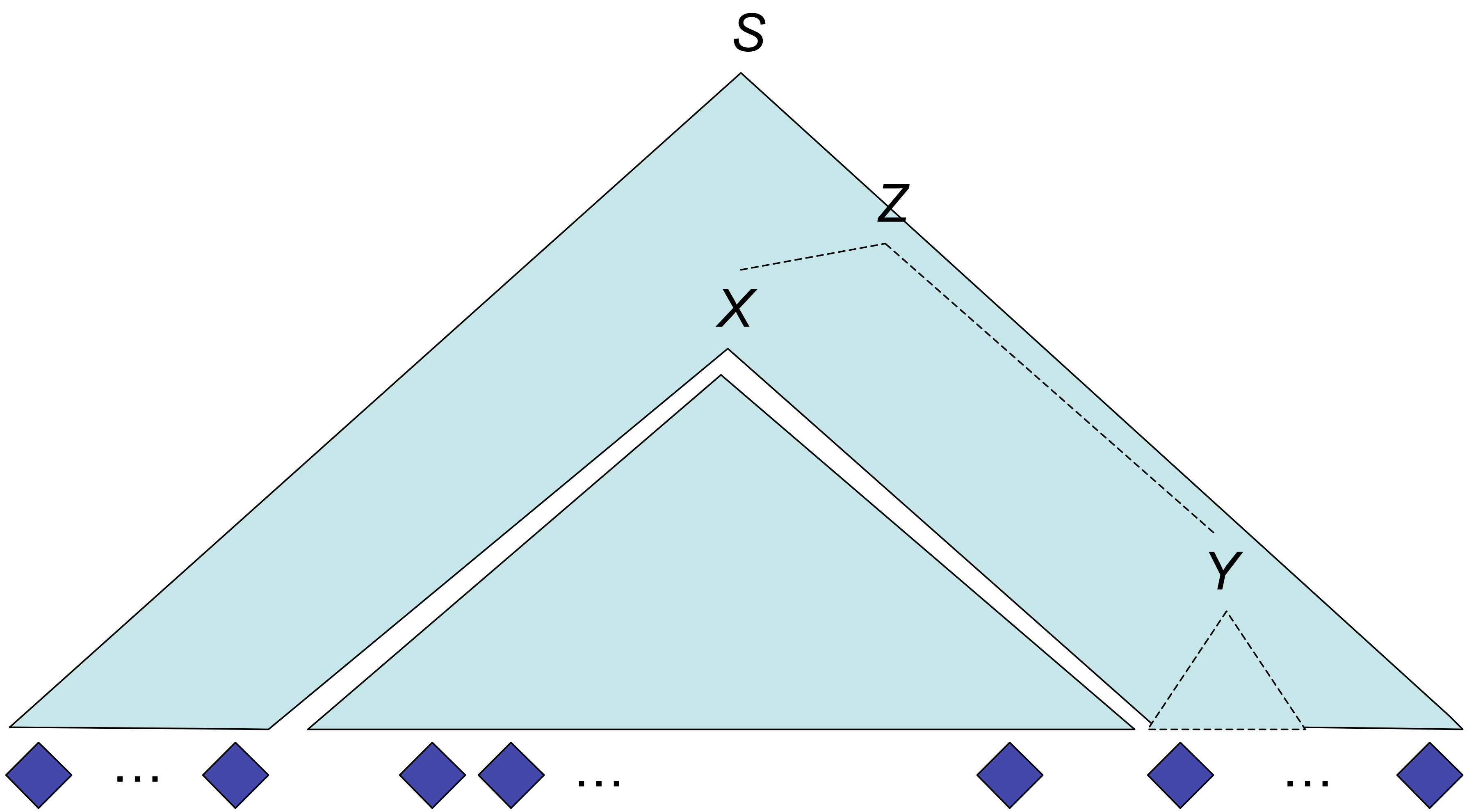
Forward and Backward Algorithms



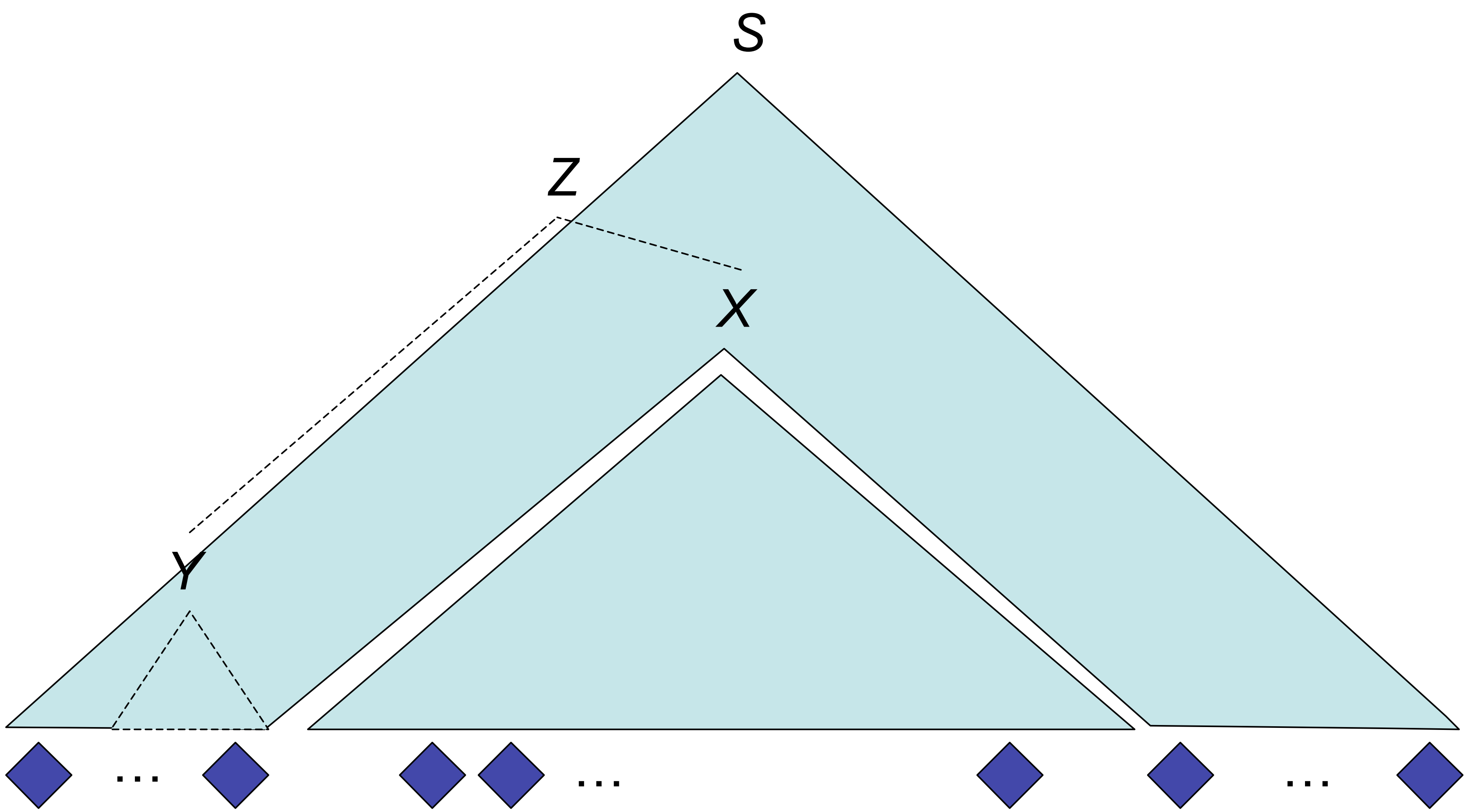
Inside and Outside Weights



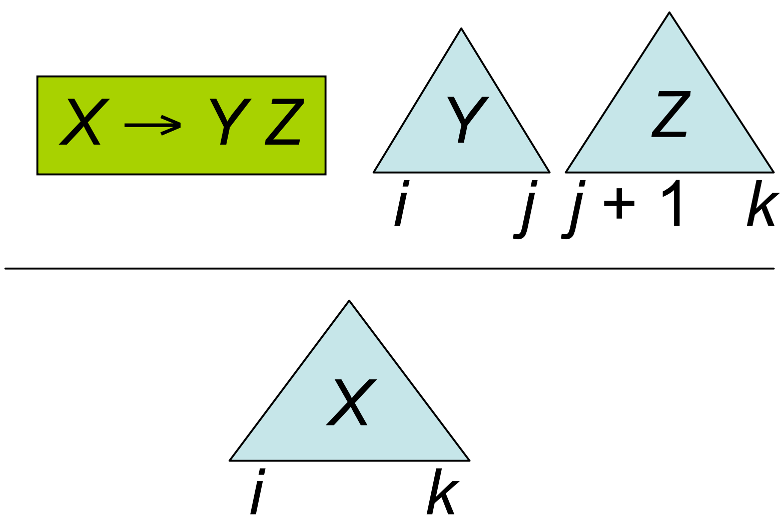
Inside and Outside Weights

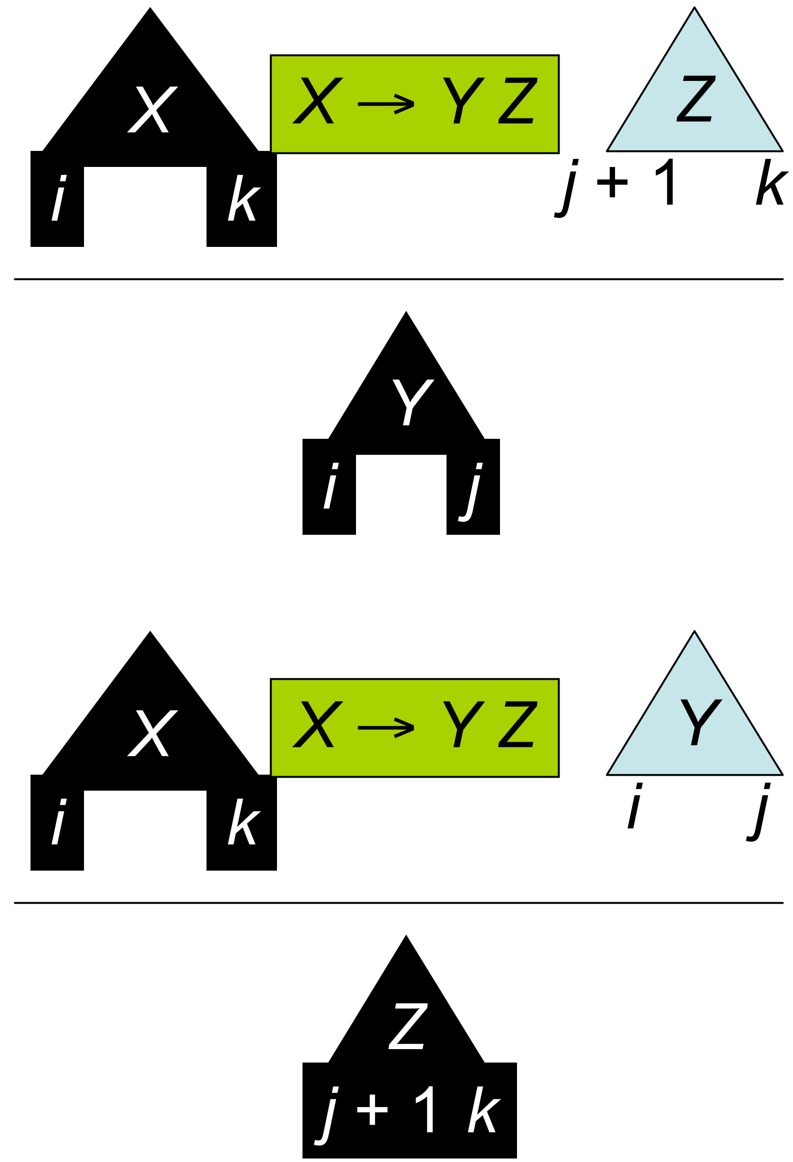


Inside and Outside Weights



Outside (CKY)





Inside and Outside Algorithms

float chart$[1..n, 1..|N|, 1..n{+}1] := 0$;
for $s := 1$ **to** n /* *start position* */
    for each rule $A \rightarrow w_s \in R$
        chart$[s, A, s{+}1] := P(A \rightarrow w_s)$;
for $l := 2$ **to** n /* *length, shortest to longest* */
    for $s := 1$ **to** $n{-}l{+}1$ /* *start position* */
        for $t := 1$ **to** $l{-}1$ /* *split length* */
            for each rule $A \rightarrow BC \in R$
                chart$[s, A, s{+}l] :=$ *chart*$[s, A, s{+}l] +$
                    $($*chart*$[s, B, s{+}t] \times$ *chart*$[s{+}t, C, s{+}l] \times P(A \rightarrow BC))$;
return *chart*$[1, S, n{+}1]$;

float outside$[1..n, 1..|N|, 1..n{+}1] := 0$;
outside$[1, S, n{+}1] := 1$;
for $l := n$ **down to** 2 /* *length, longest to shortest* */
    for $s := 1$ **to** $n{-}l{+}1$ /* *start position* */
        for $t := 1$ **to** $l{-}1$ /* *split length* */
            for each rule $A \rightarrow BC \in R$
                outside$[s, B, s{+}t] :=$ *outside*$[s, B, s{+}t] +$
                    $($*outside*$[s, A, s{+}l] \times$ *inside*$[s{+}t, C, s{+}l] \times P(A \rightarrow BC))$;
                outside$[s{+}t, C, s{+}l] :=$ *outside*$[s{+}t, C, s{+}l] +$
                    $($*outside*$[s, A, s{+}l] \times$ *inside*$[s, B, s{+}t] \times P(A \rightarrow BC))$;

Beyond Goodman (1999)

- Algorithm: carry out deduction to build the chart (i.e., fill in items with nonzero value); then compute their values.
  - Tough part: **efficient** ordering of items.
  - For Forward/Viterbi: order by position
  - For CKY: order by width
  - In general?
- Need efficient **execution strategy** for **arbitrary** programs.
  - Key idea: avoid unnecessary work and repropagation.

Unnecessary Work

- If you only want the best derivation, you don't want to build items that aren't in it!
- But you don't know which items to build until you have the best parse.

- Key idea: **agenda**.
  - Order **updates** to items' weights.
  - Roughly analogous to trading depth and breadth in **search**.

- Note: for exact inside/outside, all of the work **is** necessary!

Repropagation

- Suppose (NP, 4, 7) currently has a weight of 0.3, constructed by (DT, 4, 5) $\otimes$ (NP, 5, 7).
- Now suppose we find that **a better** way to build (NP, 4, 7): (DT, 4, 5) $\otimes$ (NNP, 5, 6) $\otimes$ (NNP, 6, 7) with value 0.31.
- Maybe now we have a better way to build (VP, 3, 7)! (Or anything else that used (NP, 4, 7).
- Have to re-build all of those consequents, and compare again, and recursively repropagate to consequents of any item whose value changes.
- May not be $O(n^3)$ anymore!

Agenda

$S \xrightarrow{.8} NP\ VP$

$NNS \xrightarrow{.0002} quitters$

$VP \xrightarrow{.6} RB\ VB$

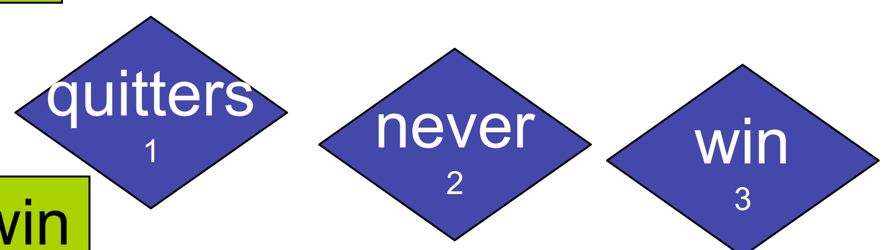
$NP \xrightarrow{.1} NNS$

$RB \xrightarrow{.04} never$

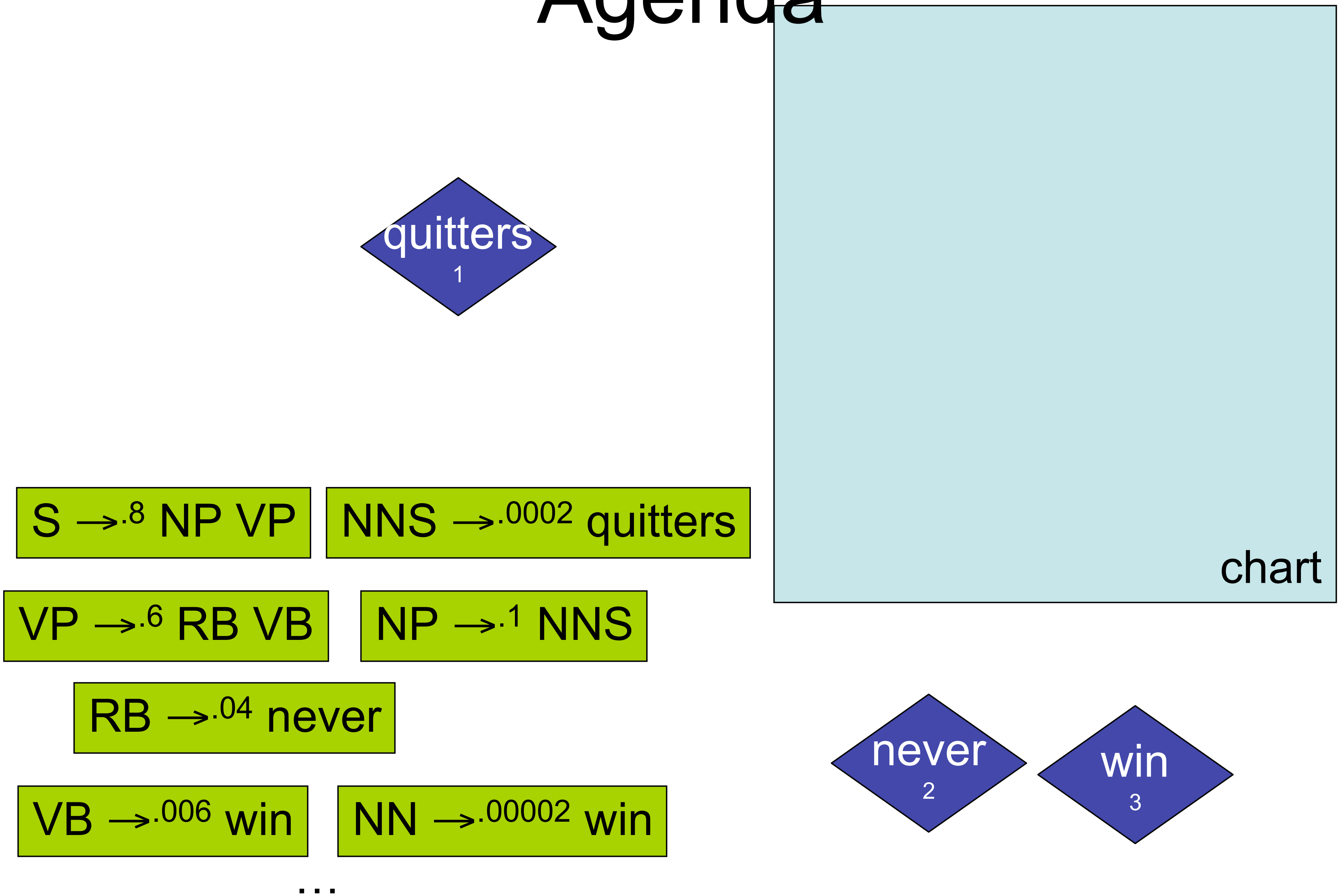
$VB \xrightarrow{.006} win$

$NN \xrightarrow{.00002} win$

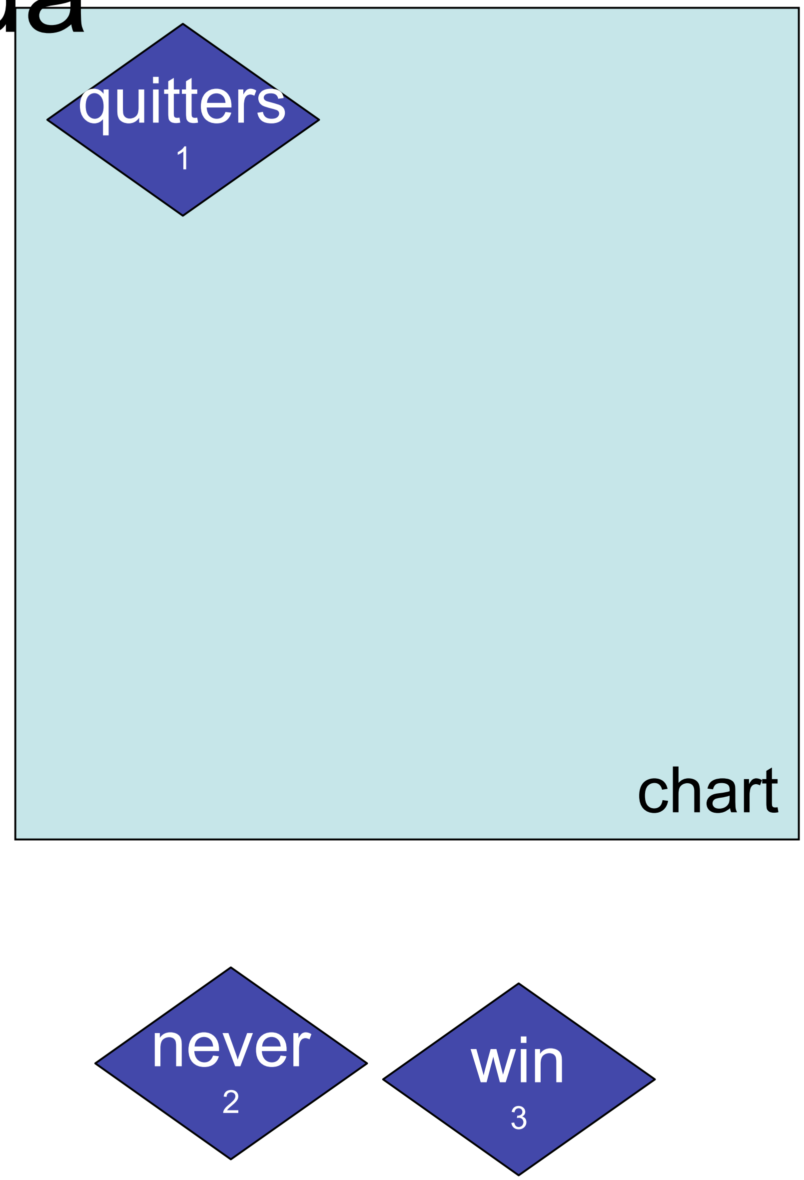
…



Agenda


quitters
1
S →.8 NP VP
NNS →.0002 quitters
VP →.6 RB VB
NP →.1 NNS
RB →.04 never
VB →.006 win
NN →.00002 win
…
chart
never
2
win
3

Agenda



S $\rightarrow^{.8}$ NP VP

NNS $\rightarrow^{.0002}$ quitters

VP $\rightarrow^{.6}$ RB VB

NP $\rightarrow^{.1}$ NNS

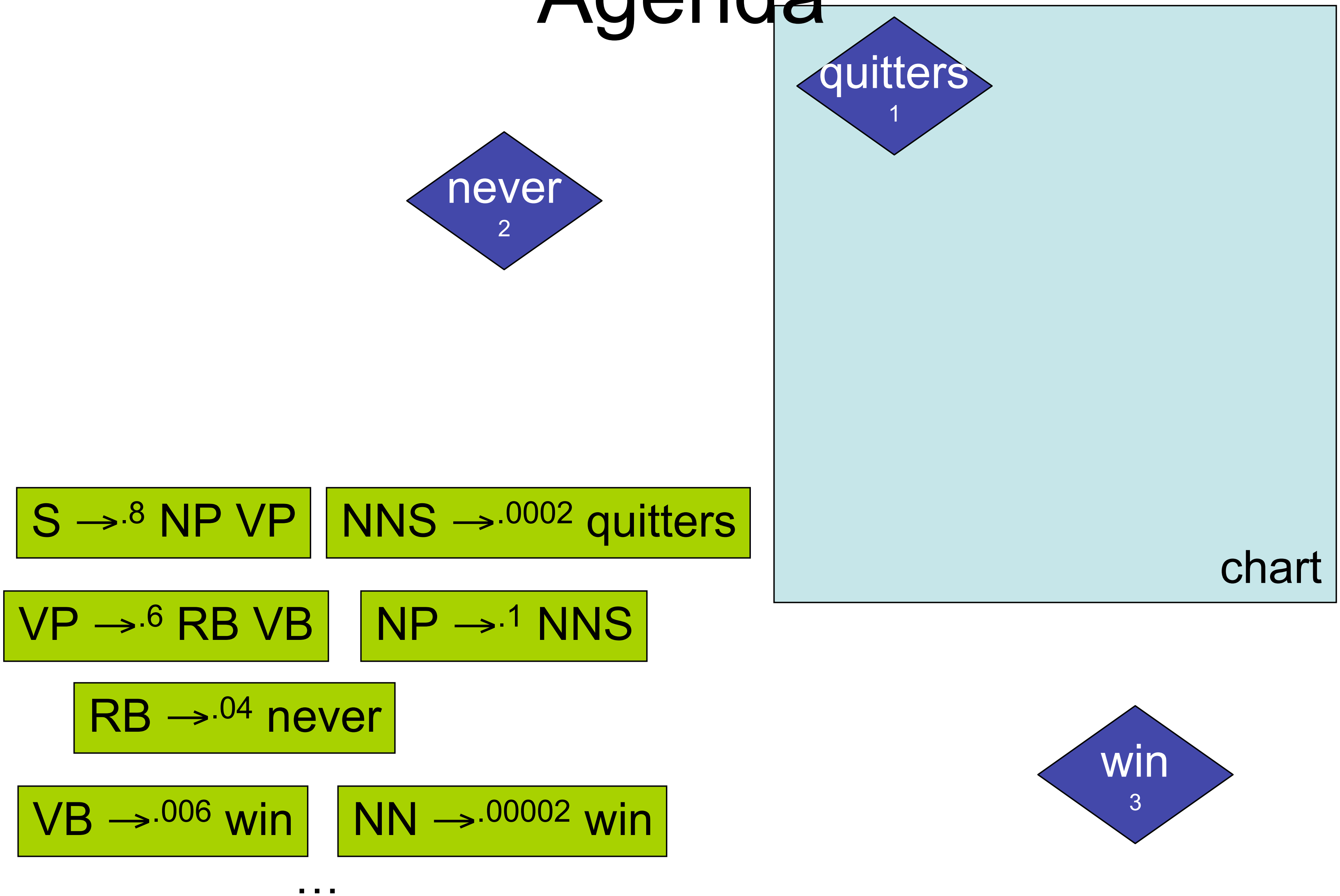
RB $\rightarrow^{.04}$ never

VB $\rightarrow^{.006}$ win

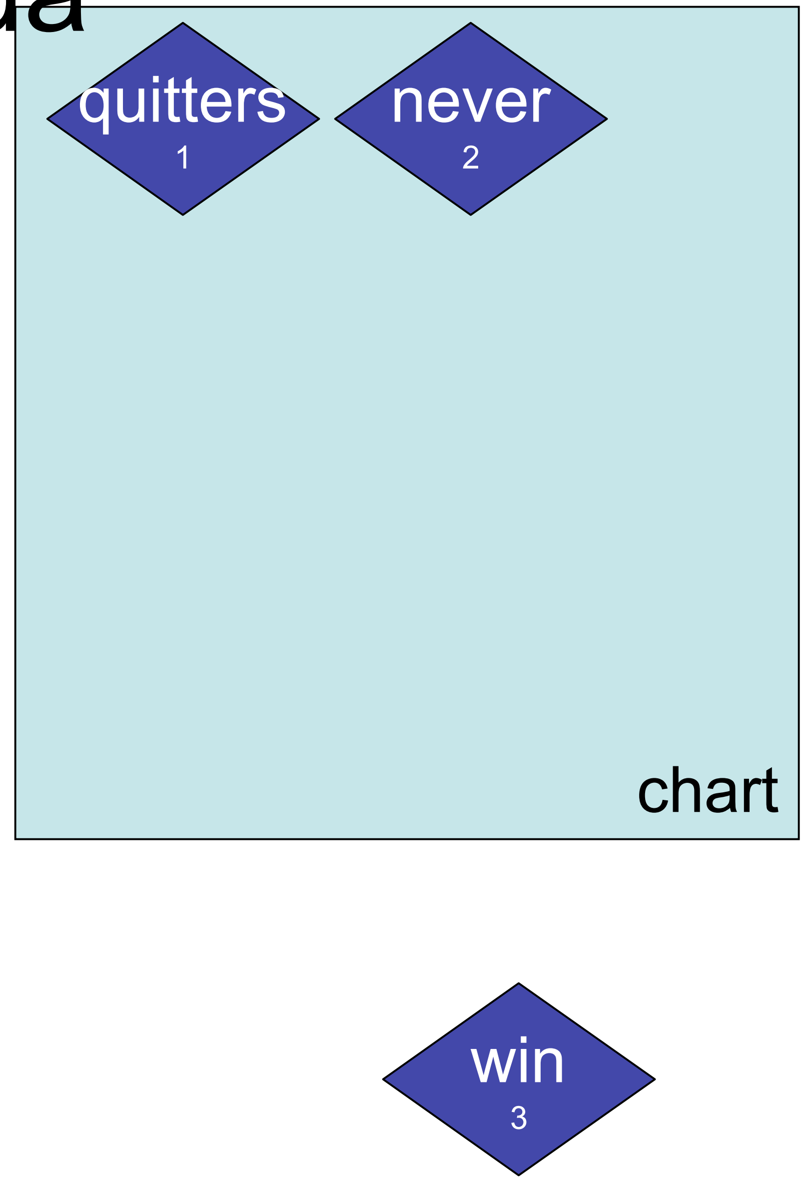
NN $\rightarrow^{.00002}$ win

…

Agenda


quitters 1
never 2
win 3
chart
S →.8 NP VP
NNS →.0002 quitters
VP →.6 RB VB
NP →.1 NNS
RB →.04 never
VB →.006 win
NN →.00002 win
…

Agenda



$S \rightarrow^{.8} NP\ VP$

$NNS \rightarrow^{.0002}$ quitters

$VP \rightarrow^{.6} RB\ VB$

$NP \rightarrow^{.1} NNS$

$RB \rightarrow^{.04}$ never

$VB \rightarrow^{.006}$ win

$NN \rightarrow^{.00002}$ win

…

Agenda

$S \rightarrow^{.8} NP\ VP$ $NNS \rightarrow^{.0002} quitters$

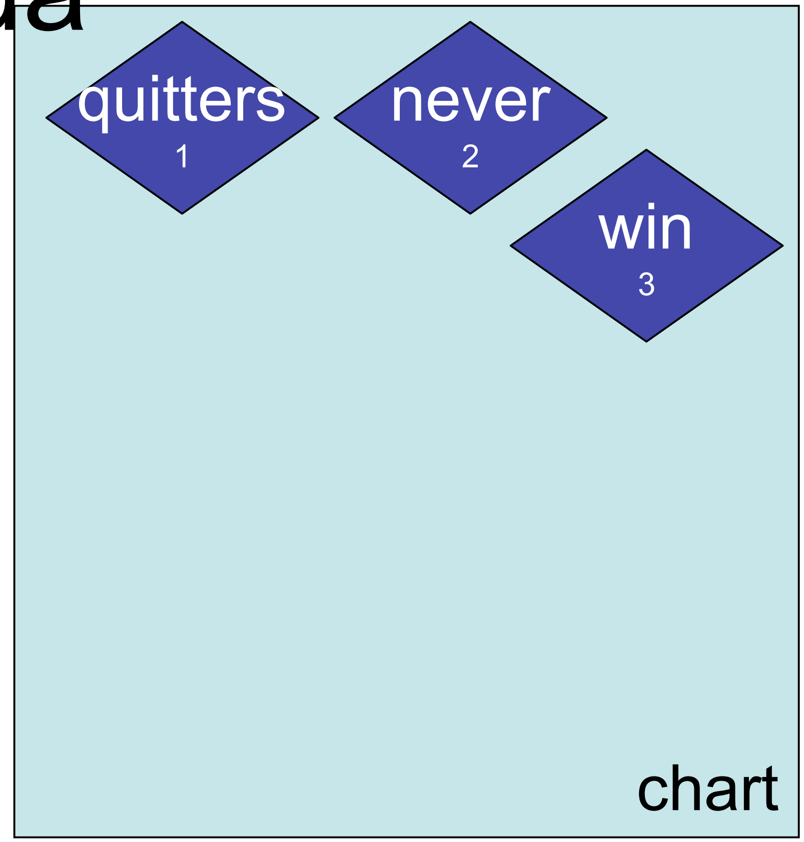
$VP \rightarrow^{.6} RB\ VB$ $NP \rightarrow^{.1} NNS$

$RB \rightarrow^{.04} never$

$VB \rightarrow^{.006} win$ $NN \rightarrow^{.00002} win$

…

Agenda



$S \rightarrow^{.8} NP\ VP$

$NNS \rightarrow^{.0002}$ quitters

$VP \rightarrow^{.6} RB\ VB$

$NP \rightarrow^{.1} NNS$

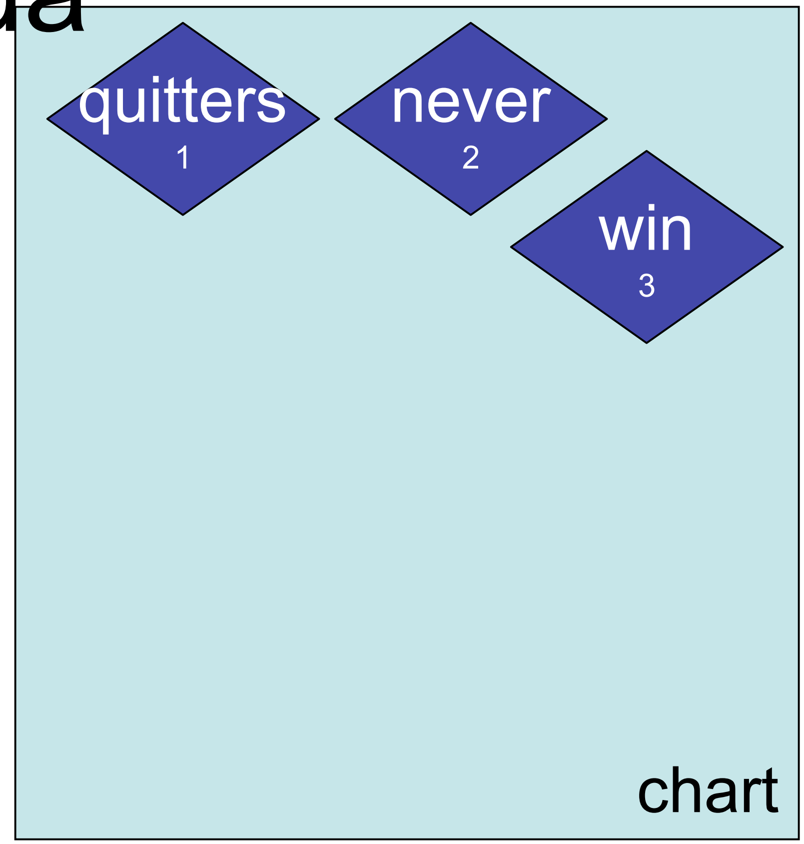
$RB \rightarrow^{.04}$ never

$VB \rightarrow^{.006}$ win

$NN \rightarrow^{.00002}$ win

…

Agenda



S $\rightarrow^{.8}$ NP VP

NNS $\rightarrow^{.0002}$ quitters

VP $\rightarrow^{.6}$ RB VB

NP $\rightarrow^{.1}$ NNS

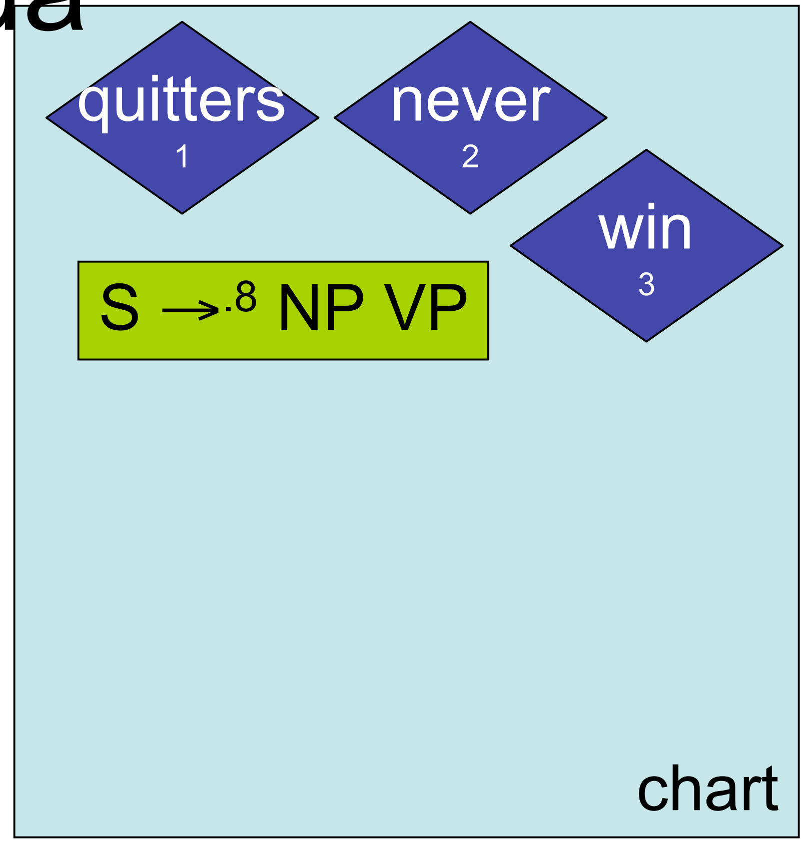
RB $\rightarrow^{.04}$ never

VB $\rightarrow^{.006}$ win

NN $\rightarrow^{.00002}$ win

…

Agenda



NNS $\rightarrow^{.0002}$ quitters

VP $\rightarrow^{.6}$ RB VB

NP $\rightarrow^{.1}$ NNS

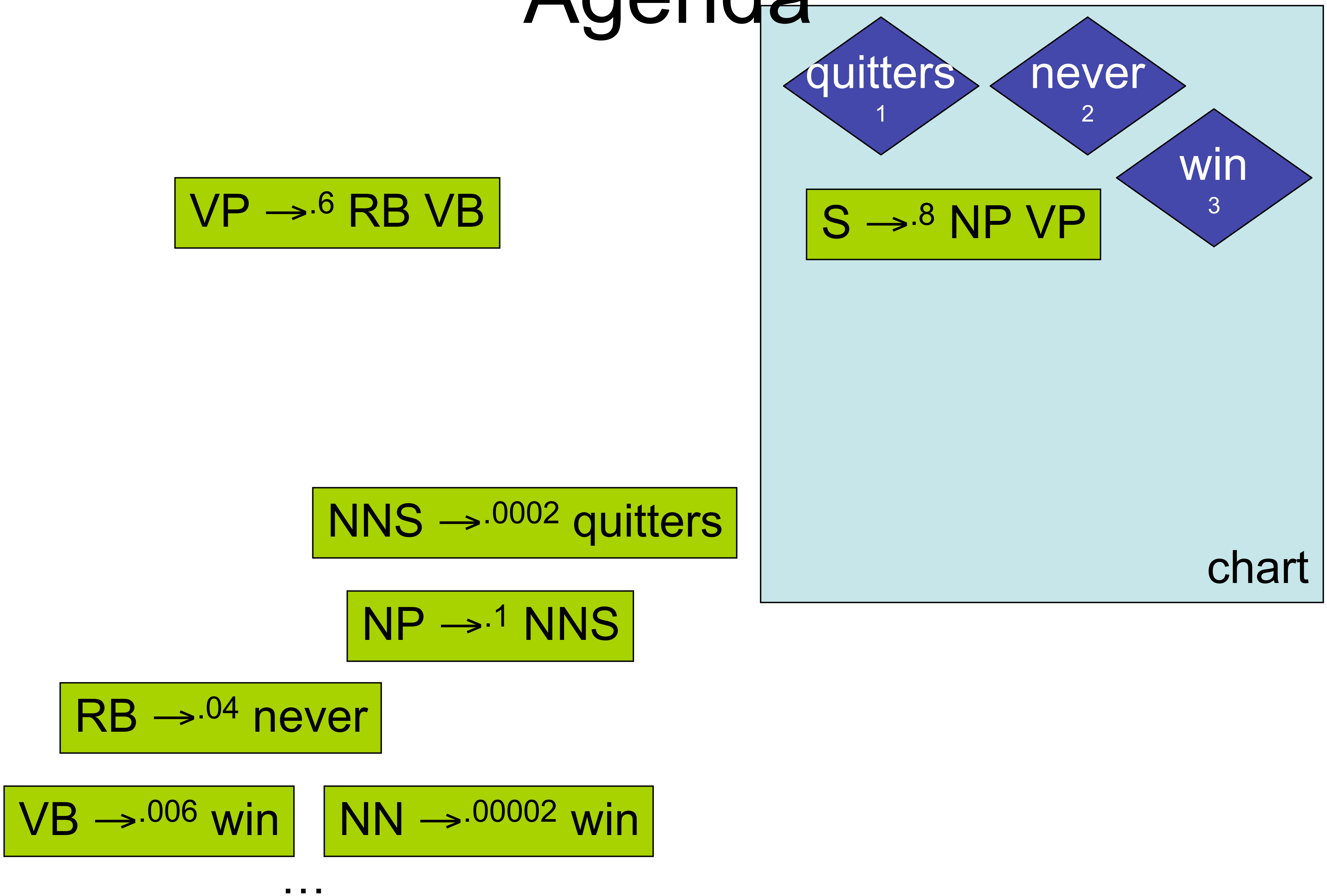
RB $\rightarrow^{.04}$ never

VB $\rightarrow^{.006}$ win

NN $\rightarrow^{.00002}$ win

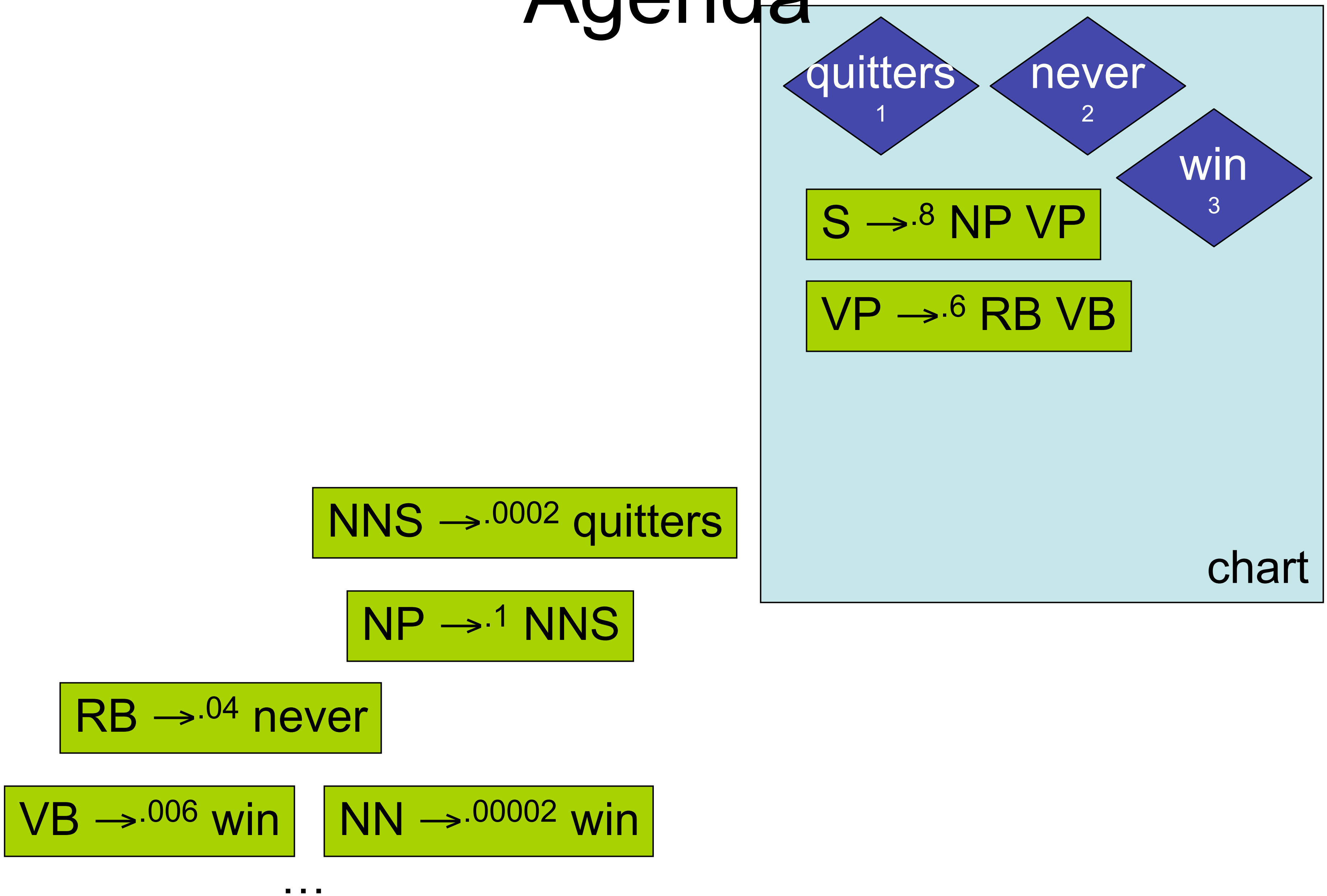
…

Agenda



quitters
1
never
2
win
3
S →.8 NP VP
chart
VP →.6 RB VB
NNS →.0002 quitters
NP →.1 NNS
RB →.04 never
VB →.006 win
NN →.00002 win
…

Agenda



quitters
1
never
2
win
3
S →.8 NP VP
VP →.6 RB VB
chart
NNS →.0002 quitters
NP →.1 NNS
RB →.04 never
VB →.006 win
NN →.00002 win
…

Agenda

NP $\rightarrow^{.1}$ NNS

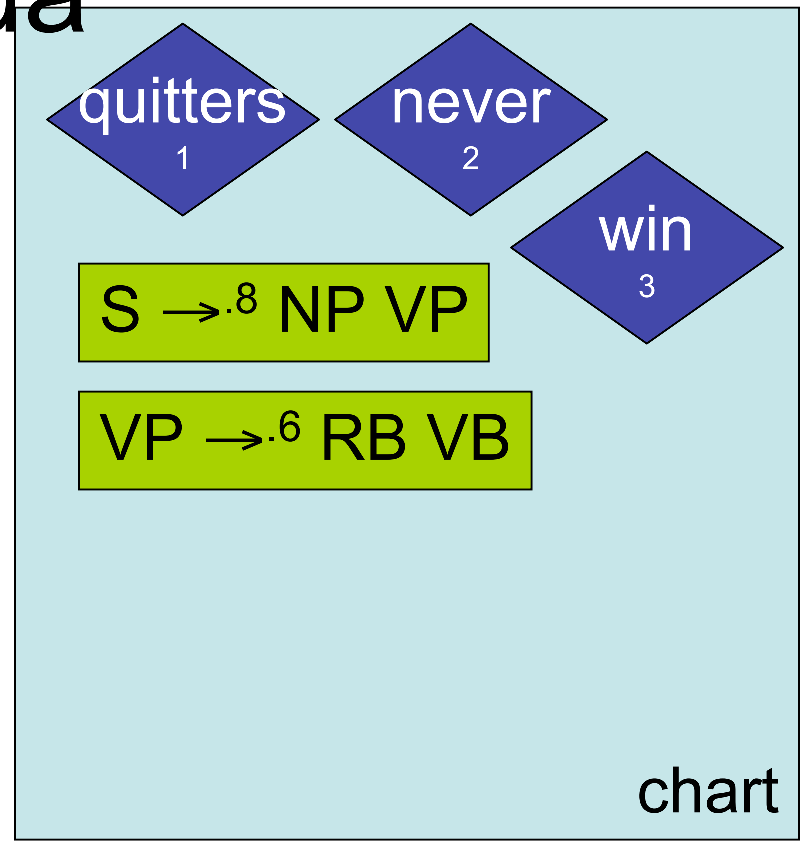
NNS $\rightarrow^{.0002}$ quitters

RB $\rightarrow^{.04}$ never

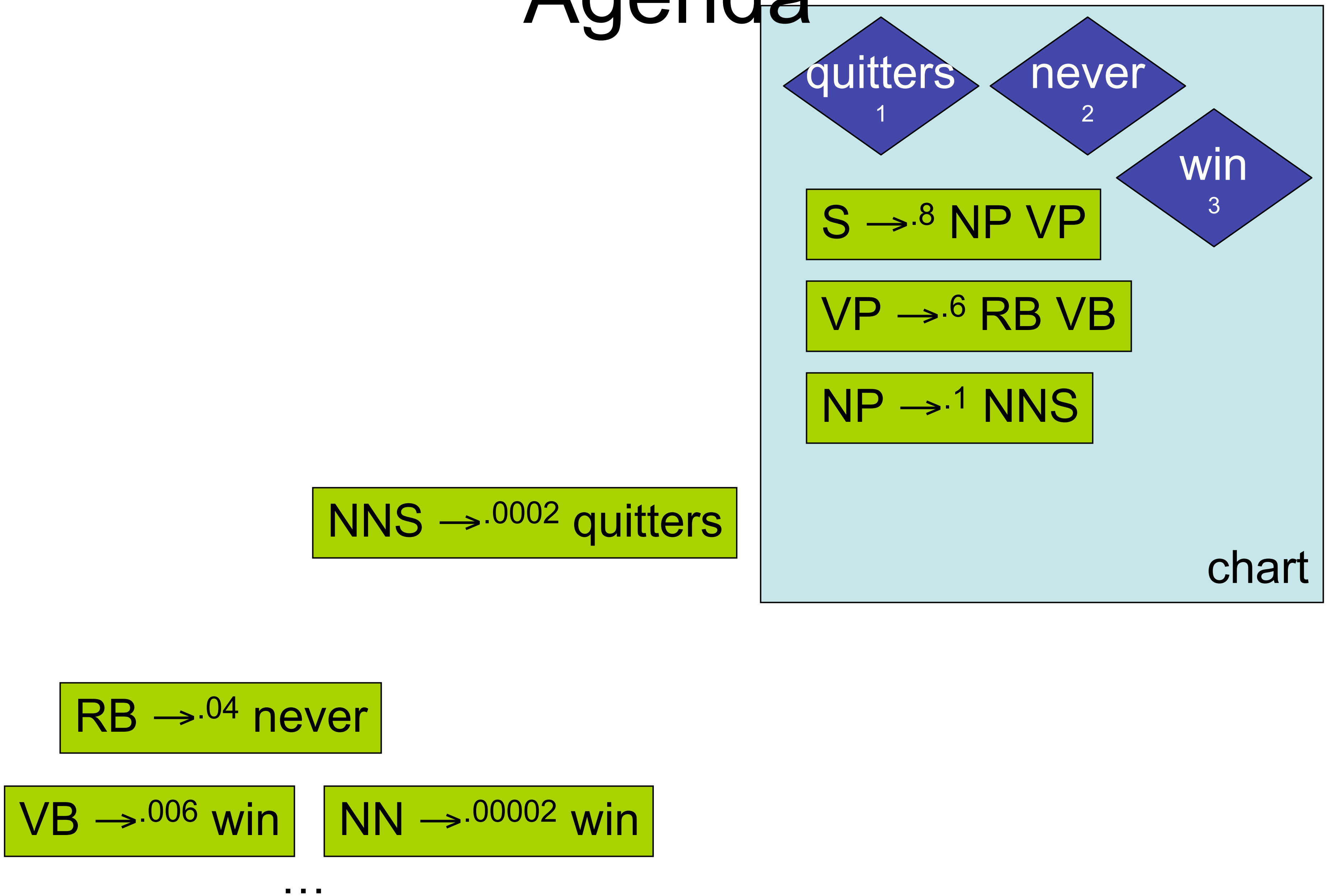
VB $\rightarrow^{.006}$ win

NN $\rightarrow^{.00002}$ win

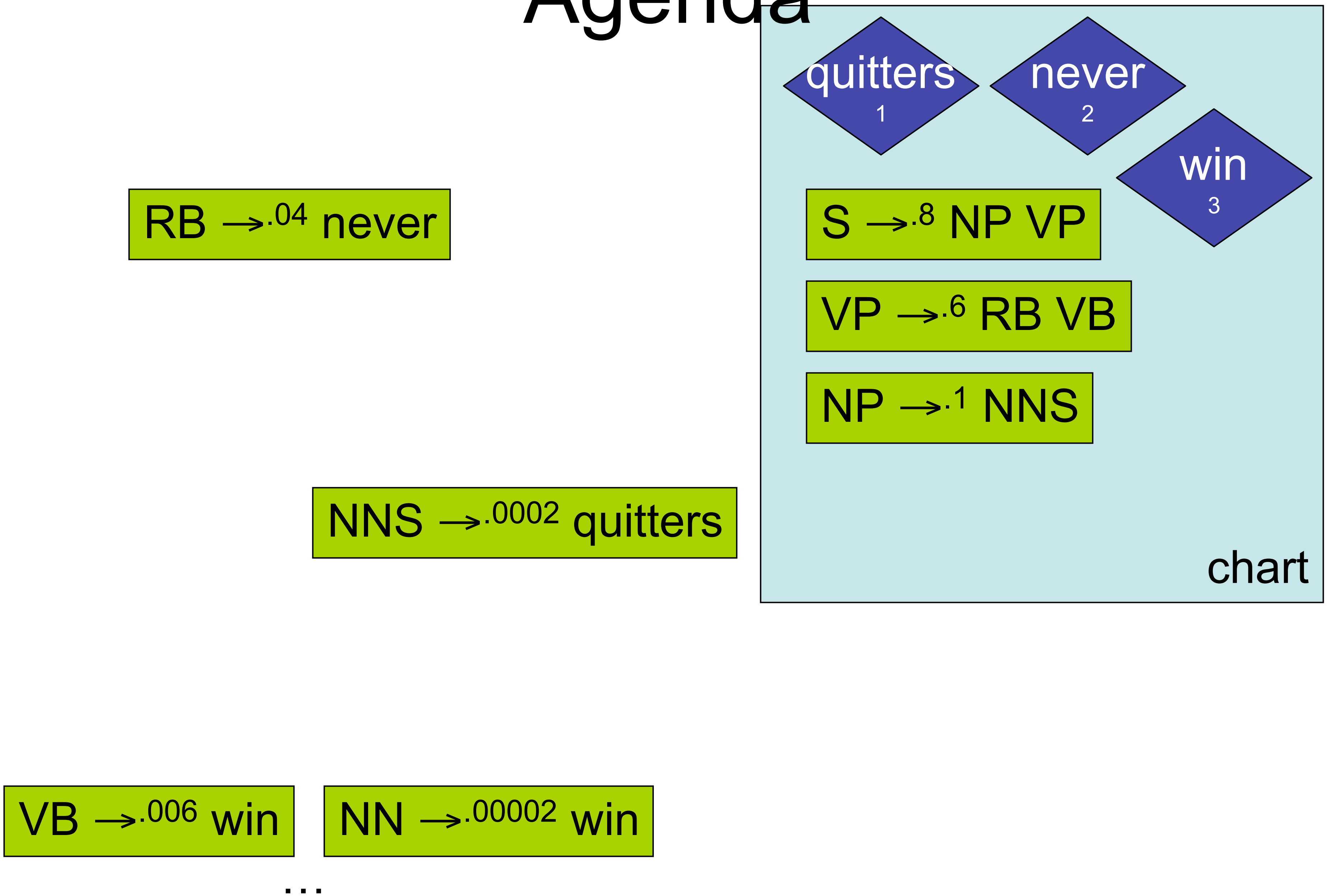
…



Agenda

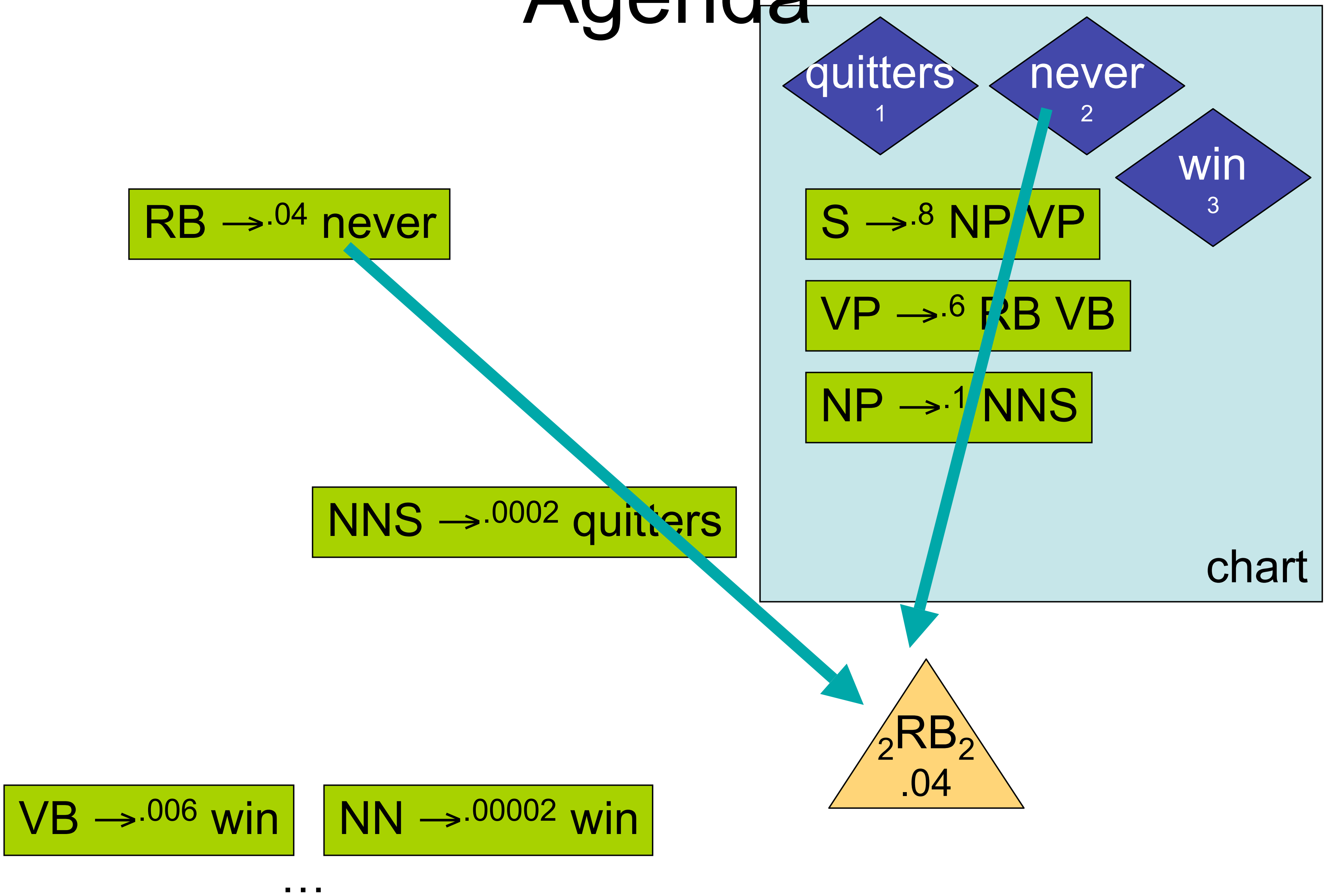

quitters
1
never
2
win
3
S →.8 NP VP
VP →.6 RB VB
NP →.1 NNS
chart
NNS →.0002 quitters
RB →.04 never
VB →.006 win
NN →.00002 win
…

Agenda



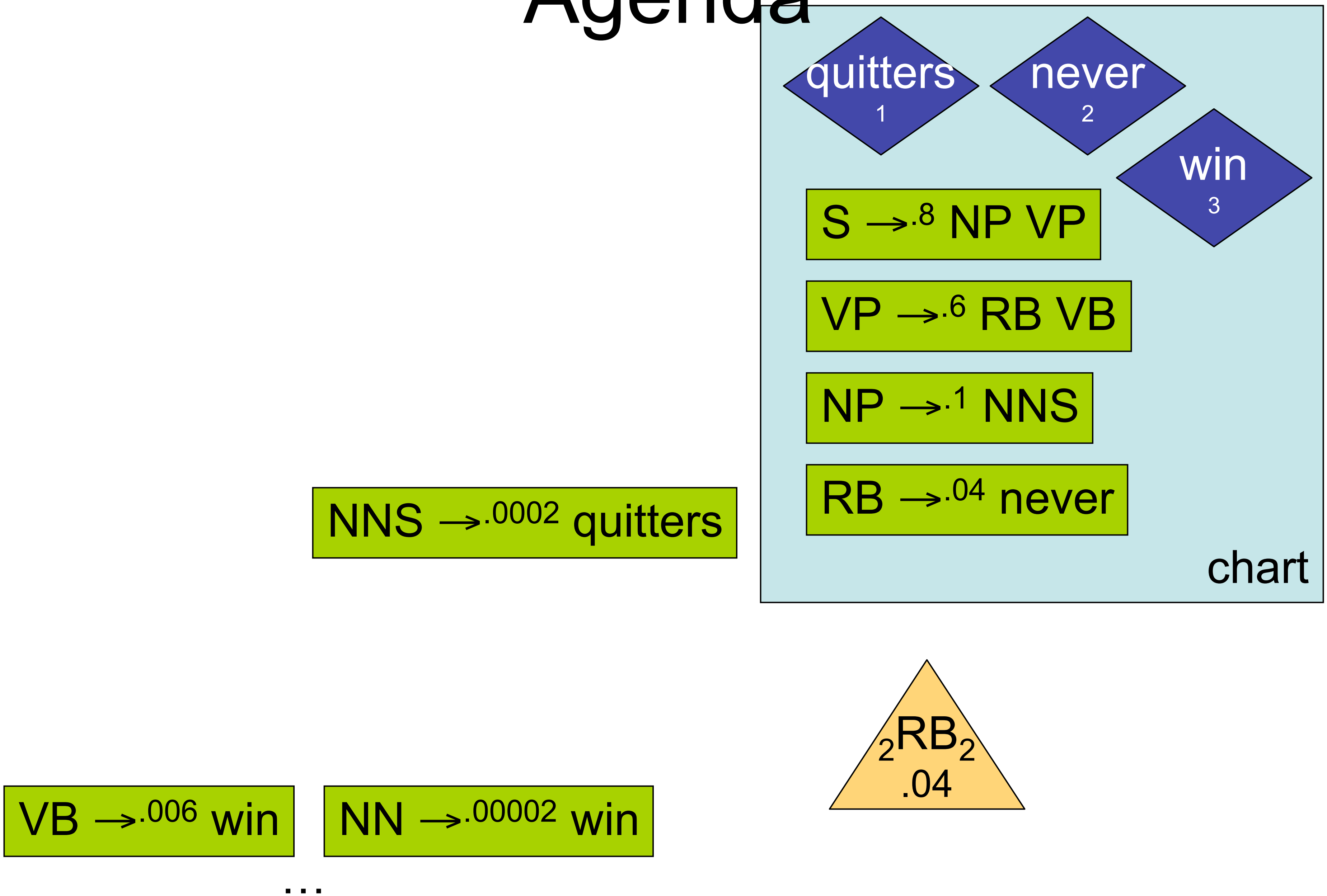
quitters 1
never 2
win 3
S →.8 NP VP
VP →.6 RB VB
NP →.1 NNS
chart
RB →.04 never
NNS →.0002 quitters
VB →.006 win
NN →.00002 win
…

Agenda



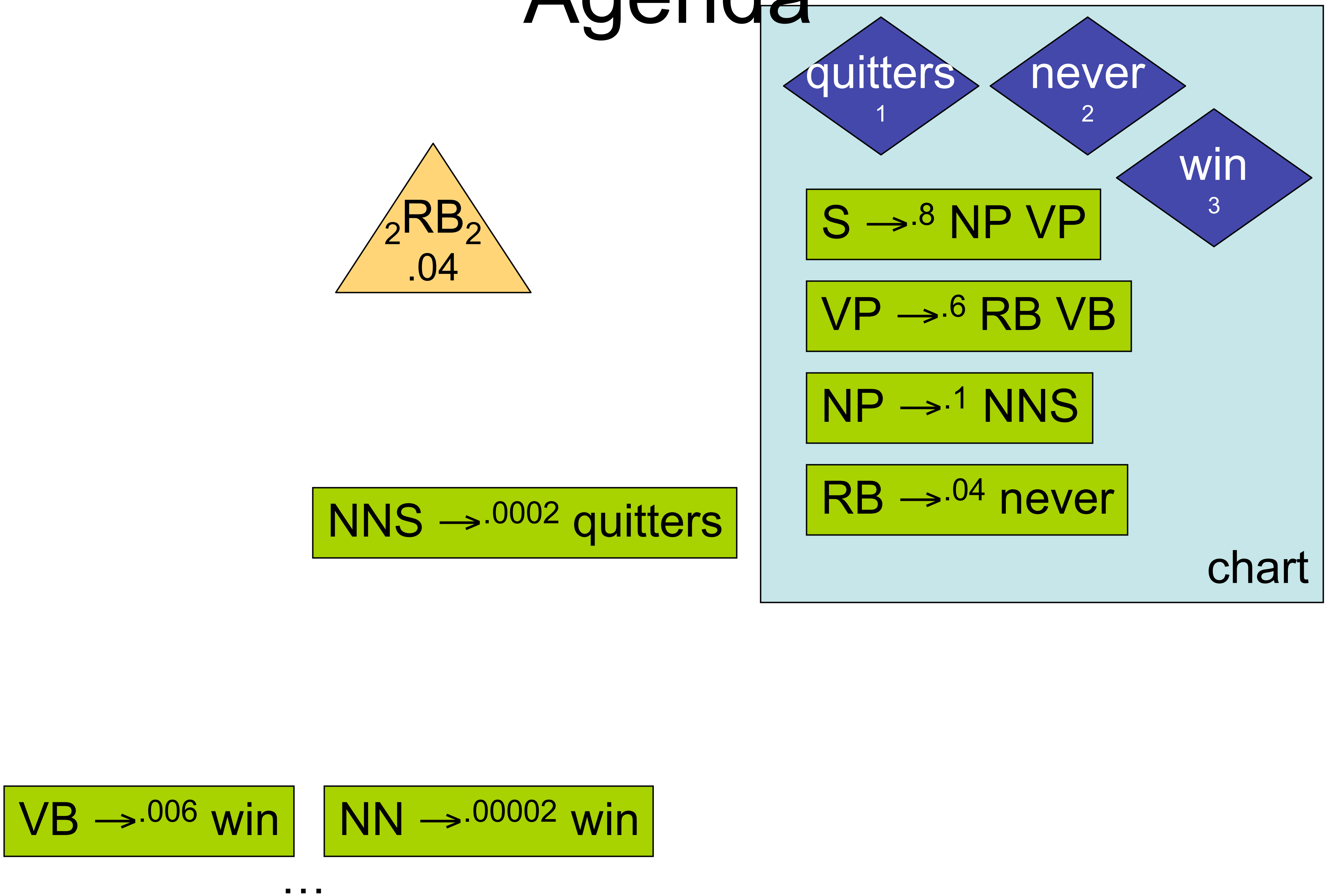
quitters 1
never 2
win 3
RB →.04 never
S →.8 NP VP
VP →.6 RB VB
NP →.1 NNS
NNS →.0002 quitters
chart
2RB2
.04
VB →.006 win
NN →.00002 win
…

Agenda



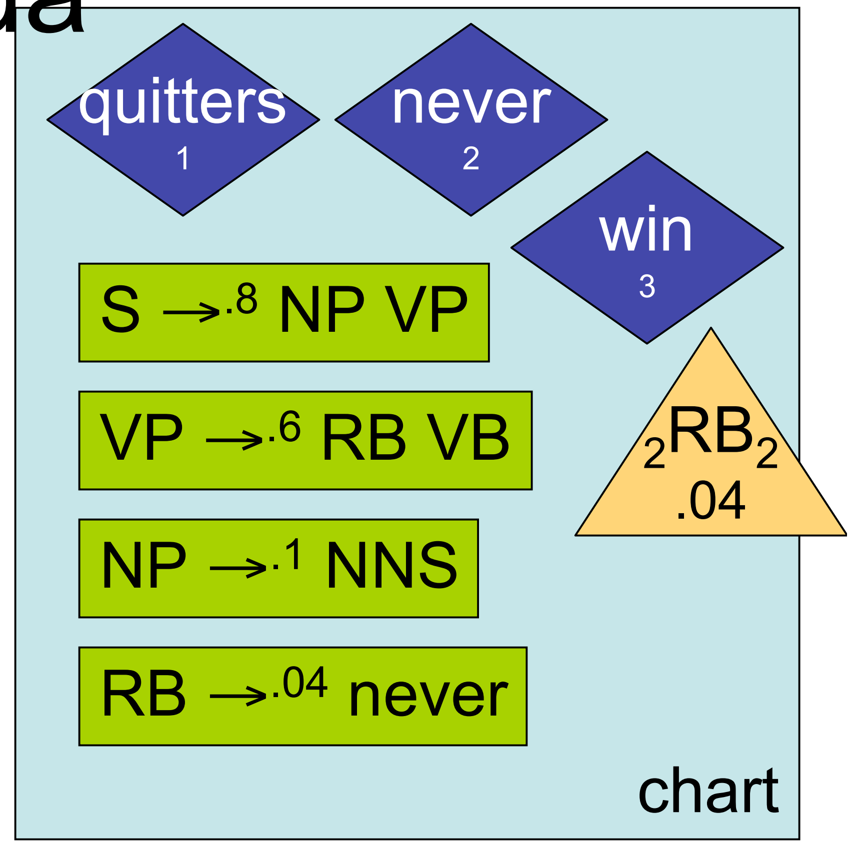
quitters 1
never 2
win 3
S →.8 NP VP
VP →.6 RB VB
NP →.1 NNS
RB →.04 never
chart
NNS →.0002 quitters
2RB2 .04
VB →.006 win
NN →.00002 win
…

Agenda



quitters
1
never
2
win
3
S →.8 NP VP
VP →.6 RB VB
NP →.1 NNS
RB →.04 never
chart
2RB2
.04
NNS →.0002 quitters
VB →.006 win
NN →.00002 win
…

Agenda



$NNS \rightarrow^{.0002}$ quitters

$VB \rightarrow^{.006}$ win

$NN \rightarrow^{.00002}$ win

…

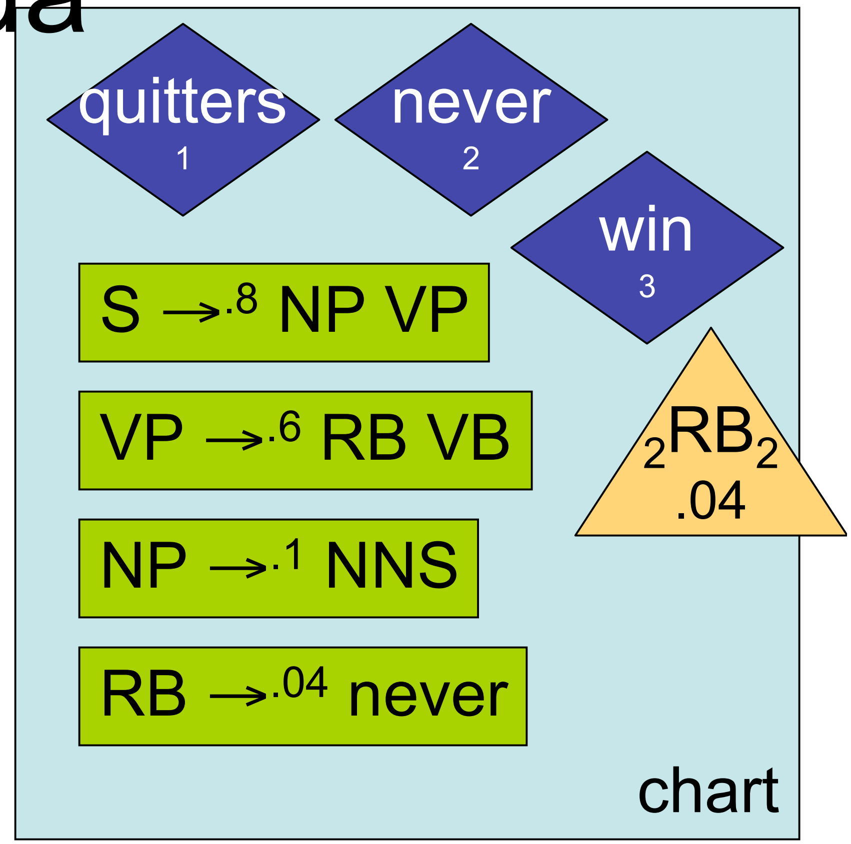
Agenda

VB $\rightarrow^{.006}$ win

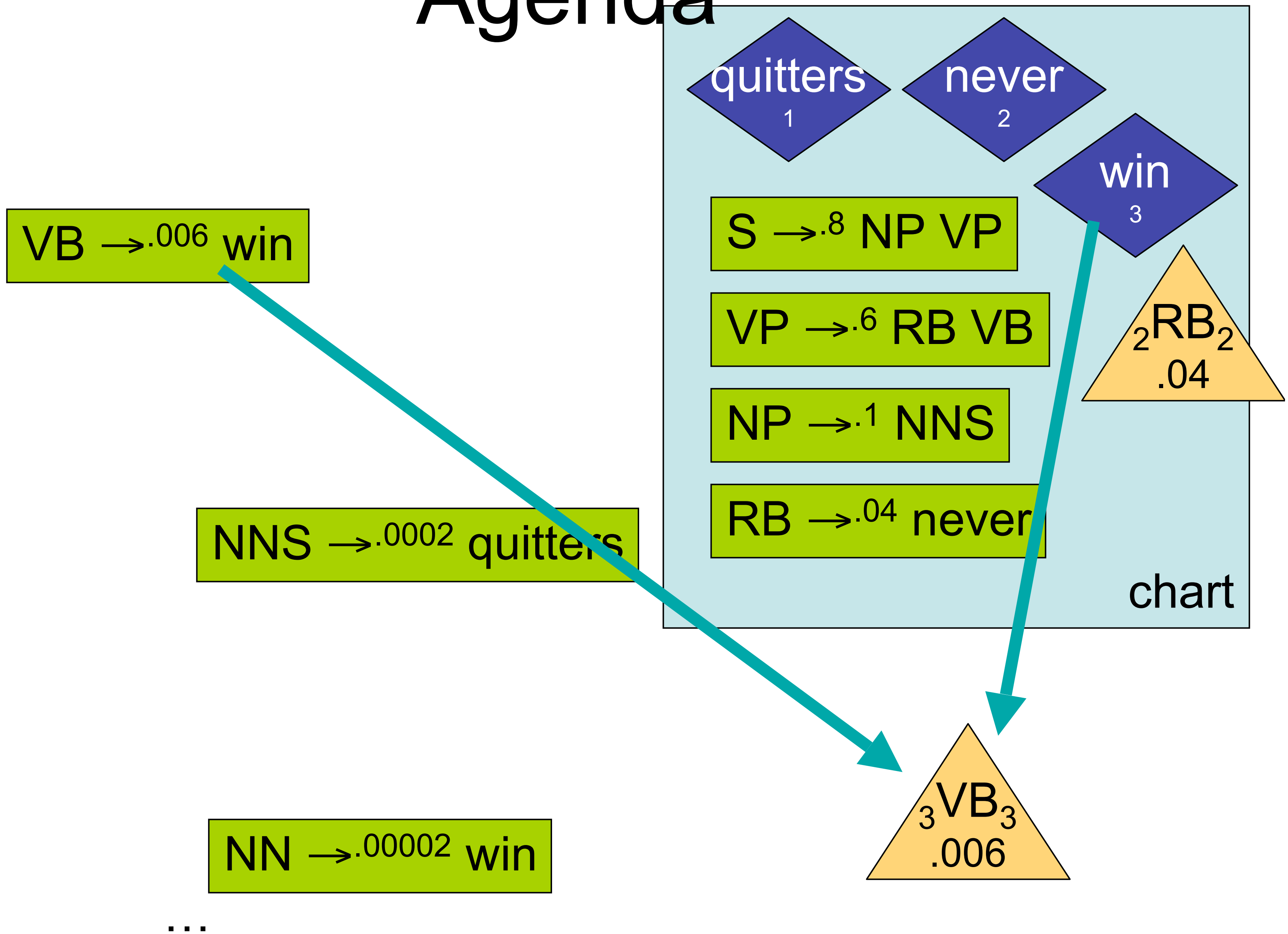
NNS $\rightarrow^{.0002}$ quitters

NN $\rightarrow^{.00002}$ win

…

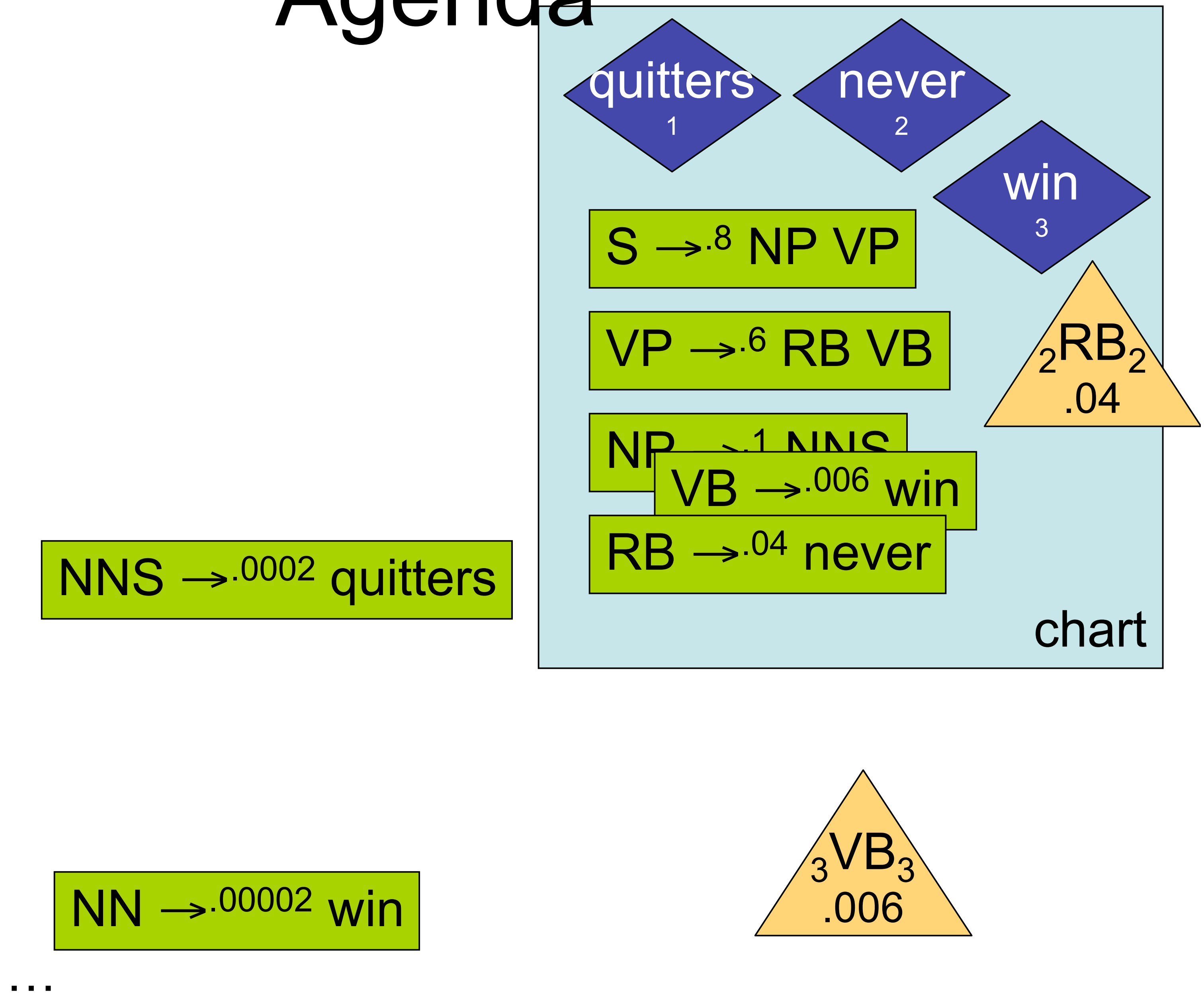


Agenda



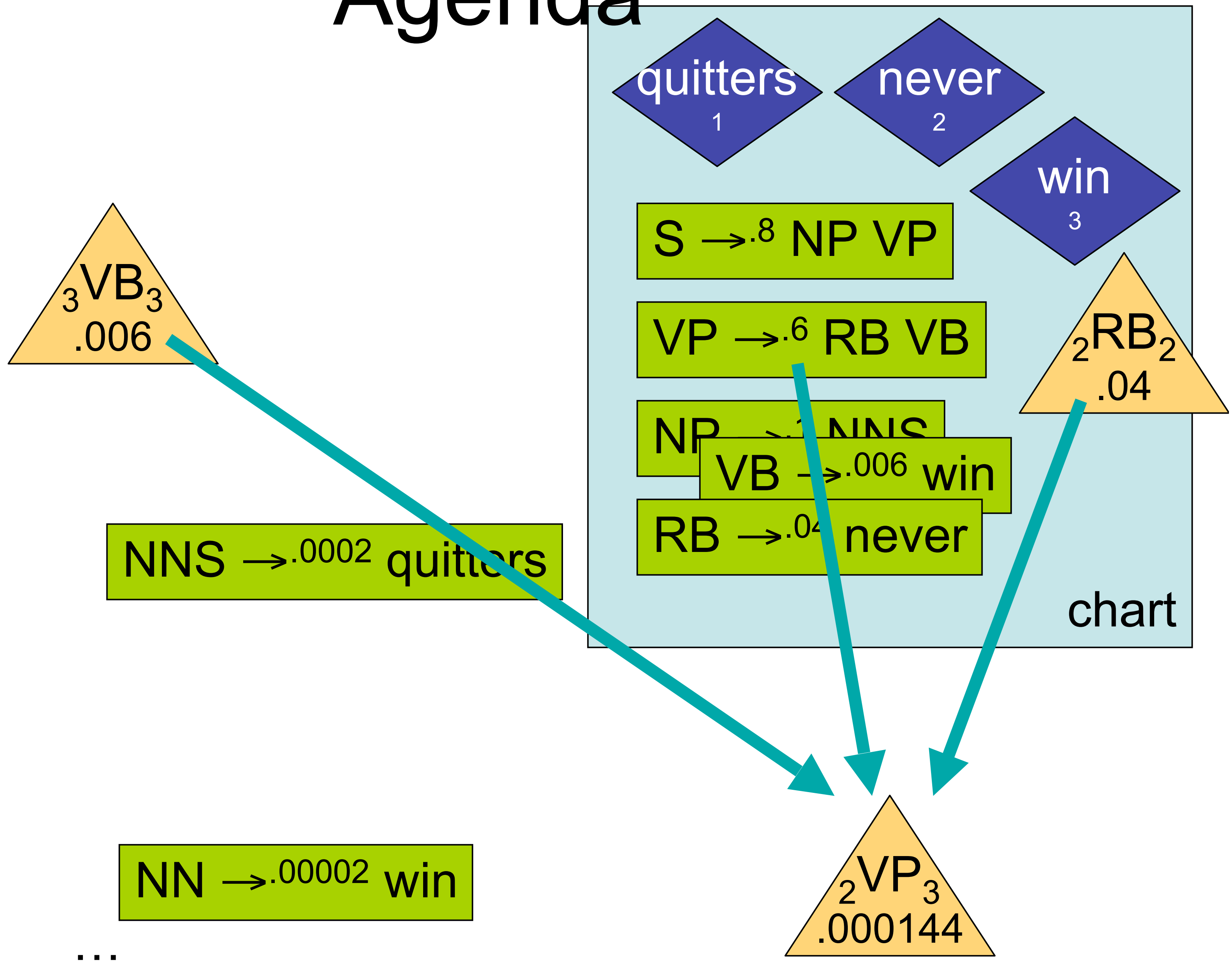
VB →.006 win
NNS →.0002 quitters
NN →.00002 win
…
quitters 1
never 2
win 3
S →.8 NP VP
VP →.6 RB VB
NP →.1 NNS
RB →.04 never
2RB2 .04
chart
3VB3 .006

Agenda



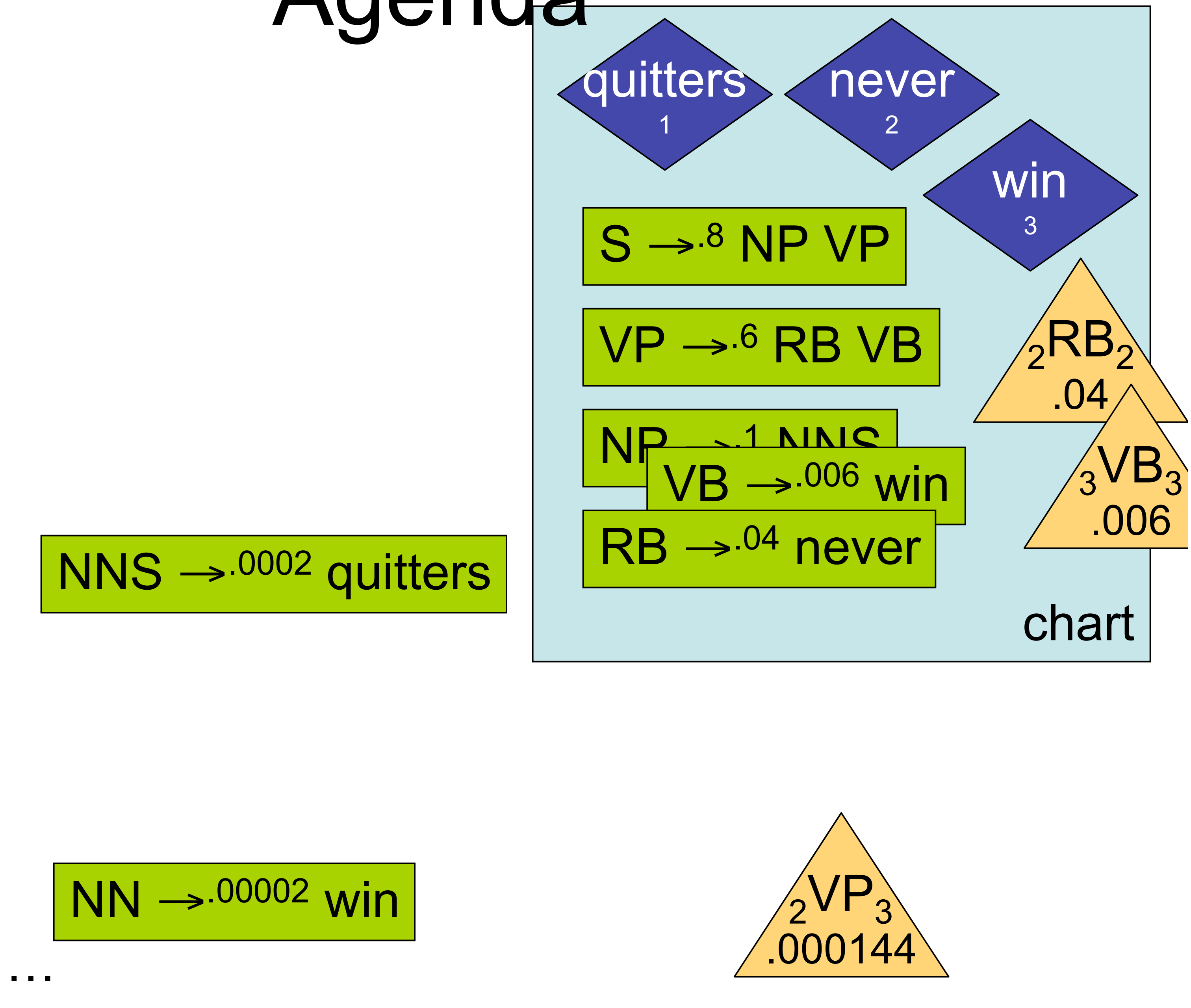
quitters
1
never
2
win
3
S →.8 NP VP
VP →.6 RB VB
NP →.1 NNS
VB →.006 win
RB →.04 never
2RB2
.04
chart
NNS →.0002 quitters
NN →.00002 win
…
3VB3
.006

Agenda



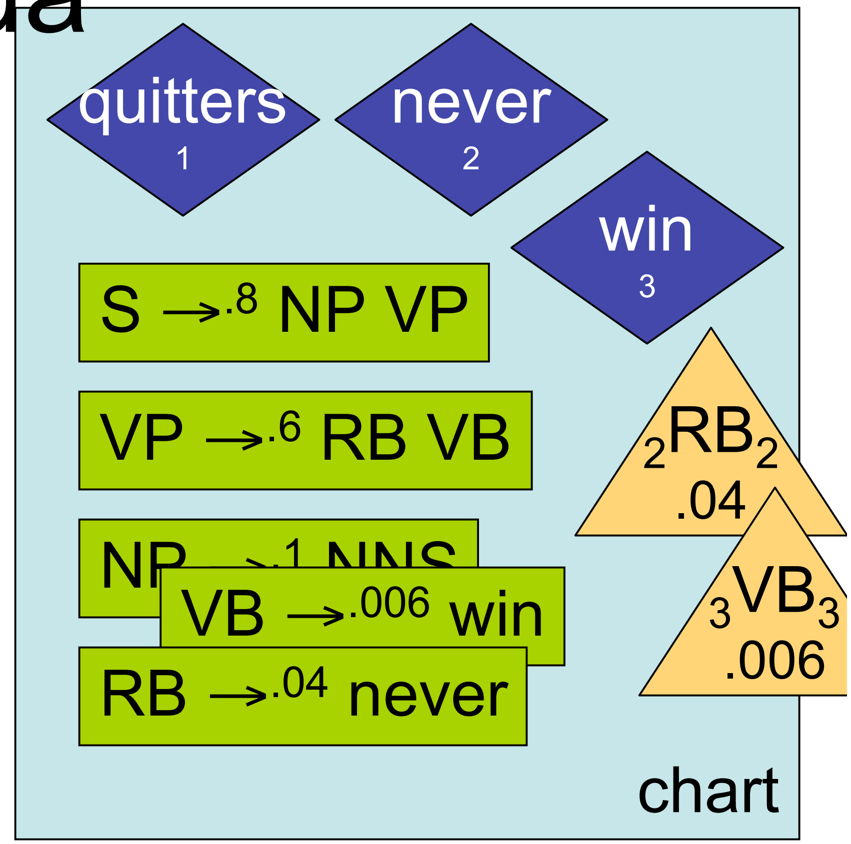
quitters
1
never
2
win
3
S →.8 NP VP
VP →.6 RB VB
VB →.006 win
RB → never
chart
$_3VB_3$
.006
$_2RB_2$
.04
NNS →.0002 quitters
NN →.00002 win
…
$_2VP_3$
.000144

Agenda


quitters
1
never
2
win
3
S →.8 NP VP
VP →.6 RB VB
NP →1 NNS
VB →.006 win
RB →.04 never
2RB2
.04
3VB3
.006
chart
NNS →.0002 quitters
NN →.00002 win
…
2VP3
.000144

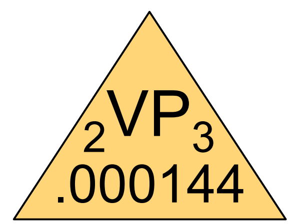
Agenda

NNS $\rightarrow^{.0002}$ quitters

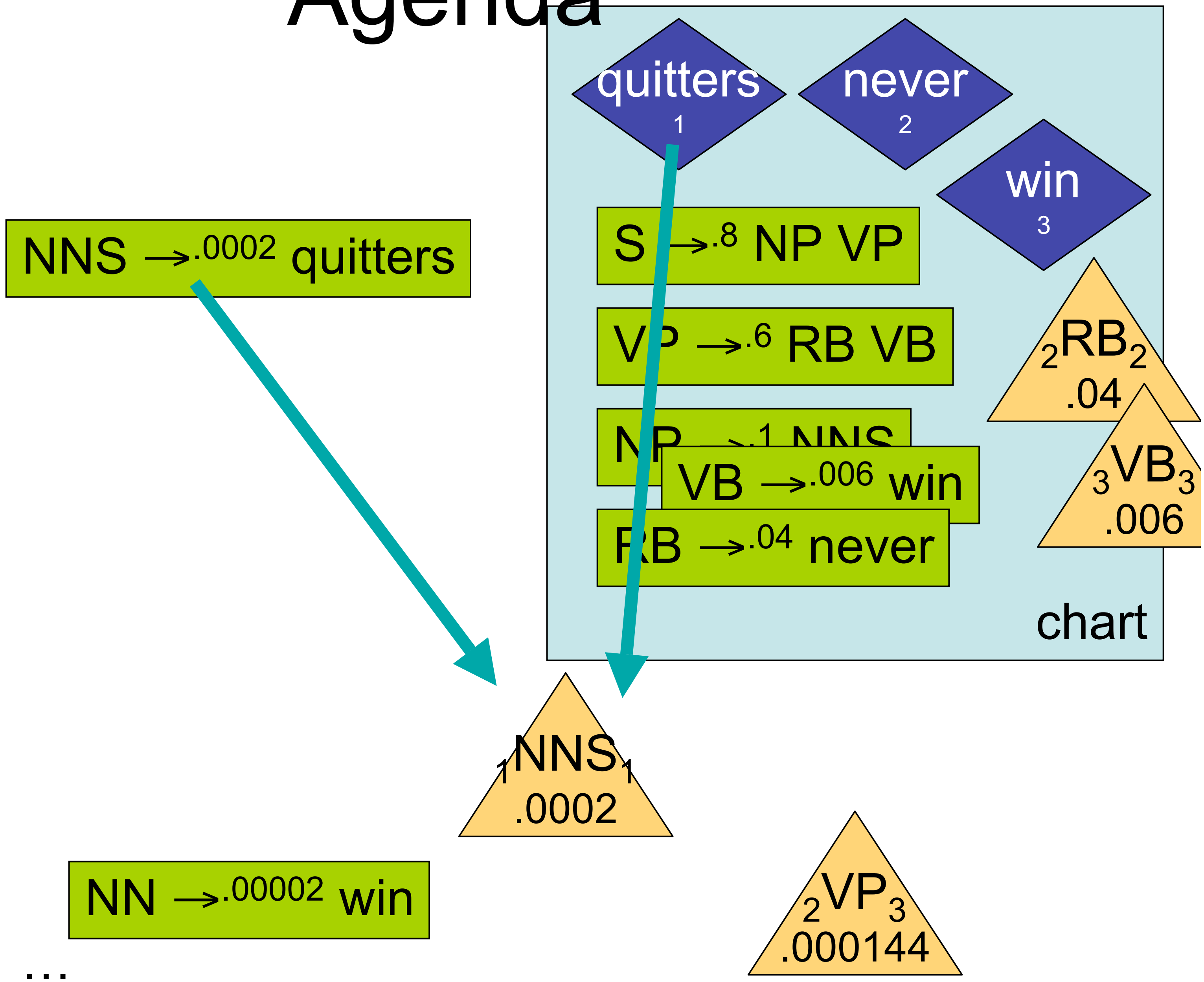


NN $\rightarrow^{.00002}$ win

…

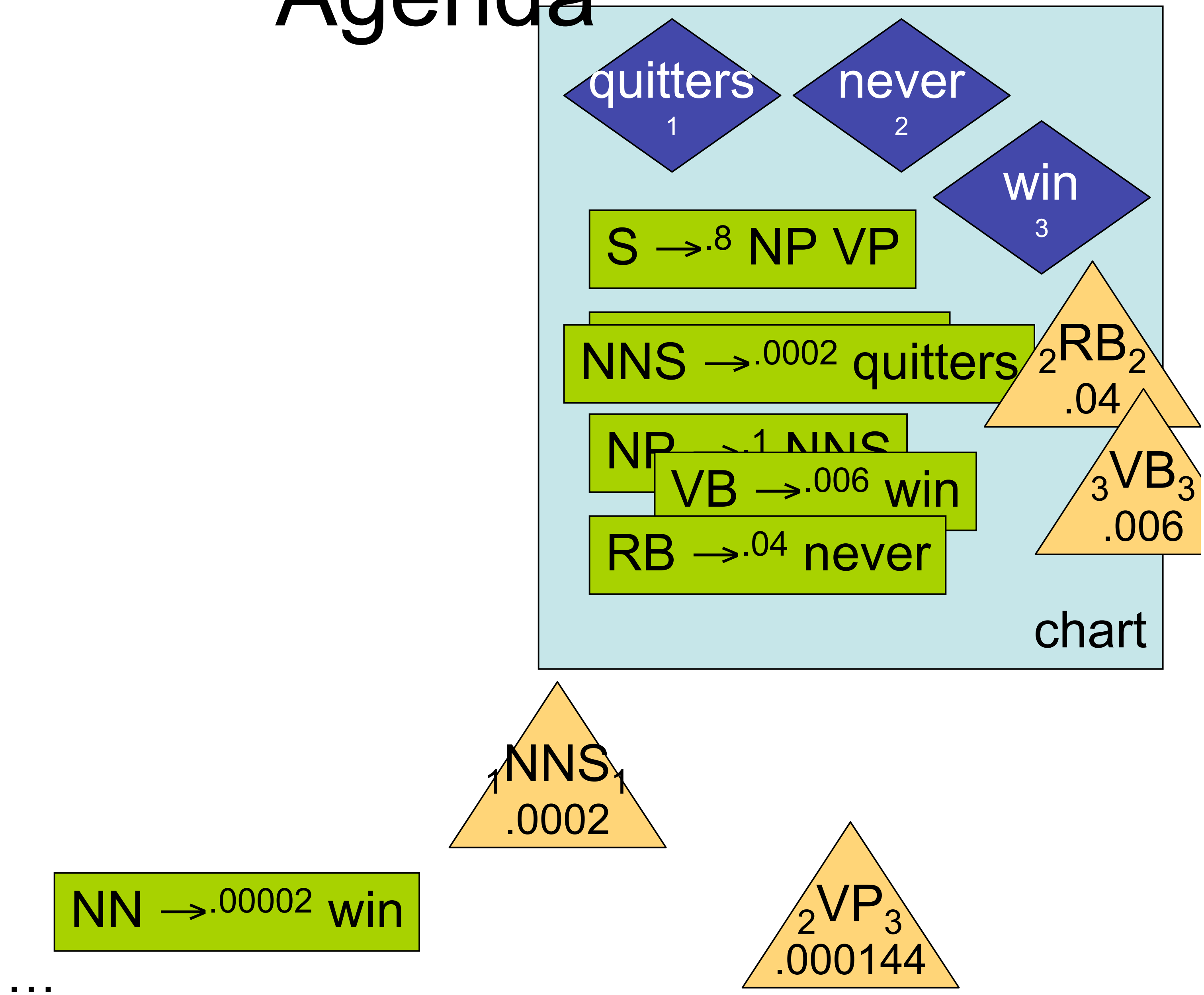


Agenda



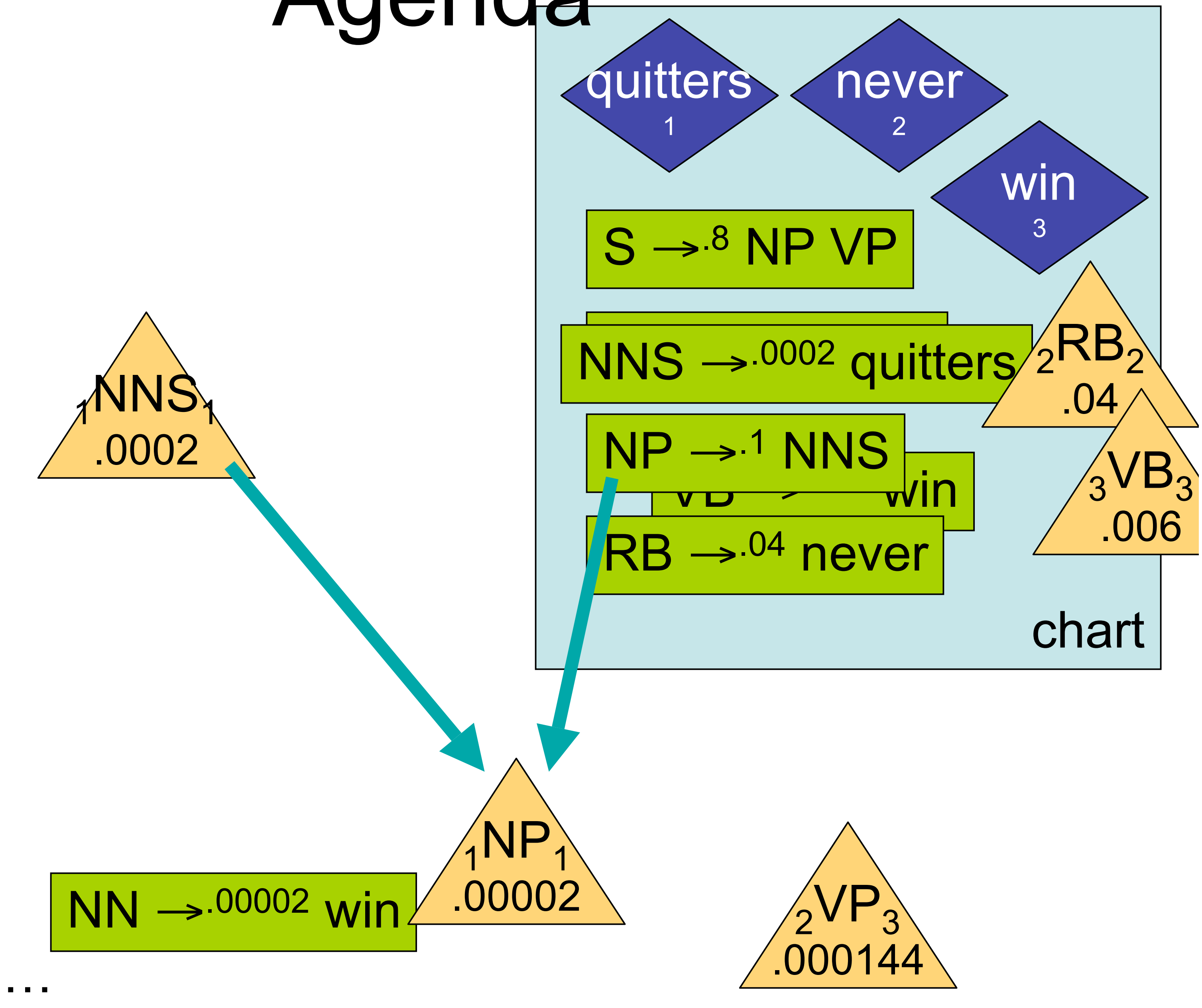
quitters 1
never 2
win 3
S →.8 NP VP
VP →.6 RB VB
NP →.1 NNS
VB →.006 win
RB →.04 never
2RB2 .04
3VB3 .006
chart
NNS →.0002 quitters
1NNS1 .0002
NN →.00002 win
2VP3 .000144
…

Agenda



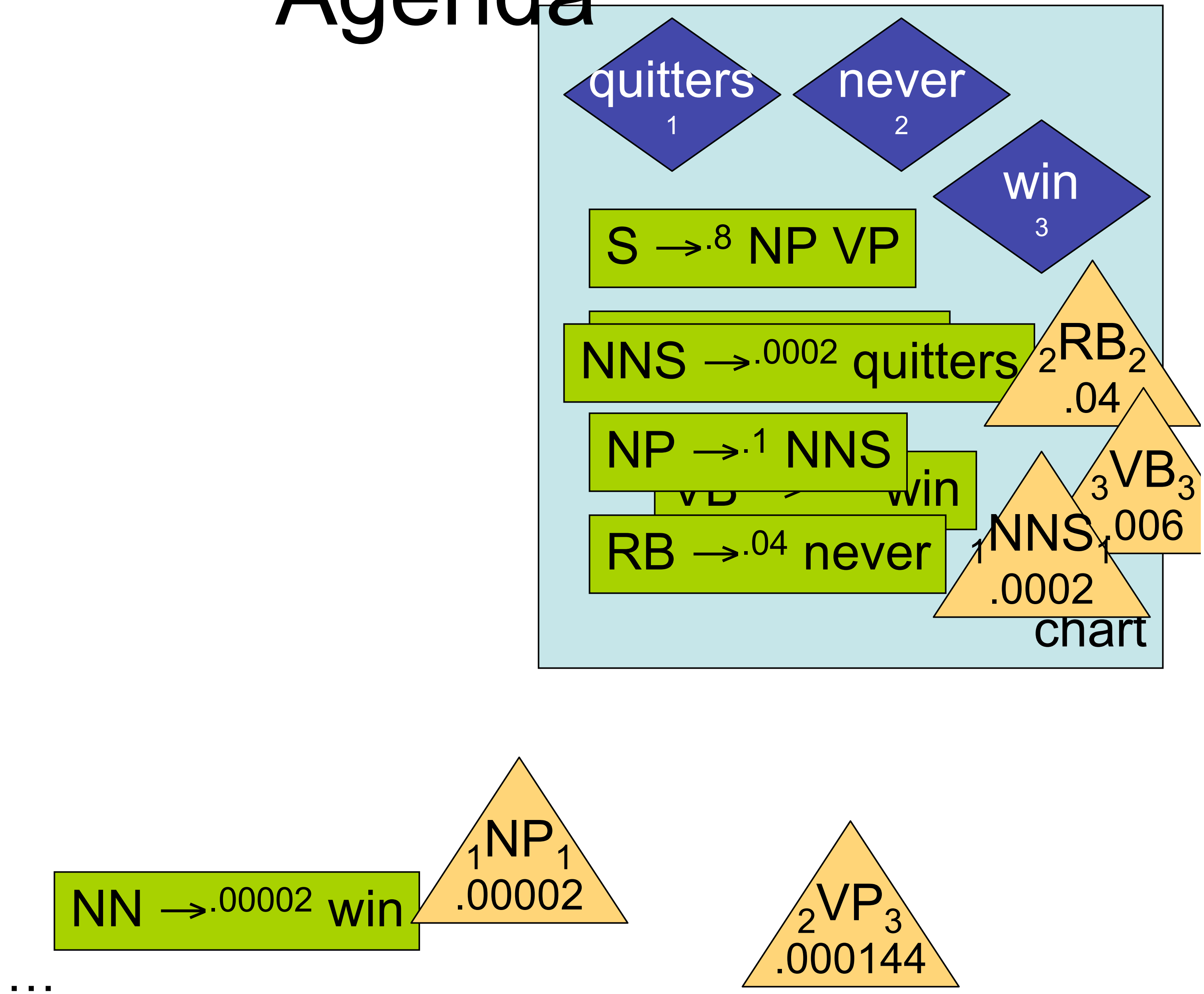
quitters 1
never 2
win 3
S →.8 NP VP
NNS →.0002 quitters
VB →.006 win
RB →.04 never
2RB2 .04
3VB3 .006
chart
1NNS1 .0002
NN →.00002 win
2VP3 .000144
…

Agenda

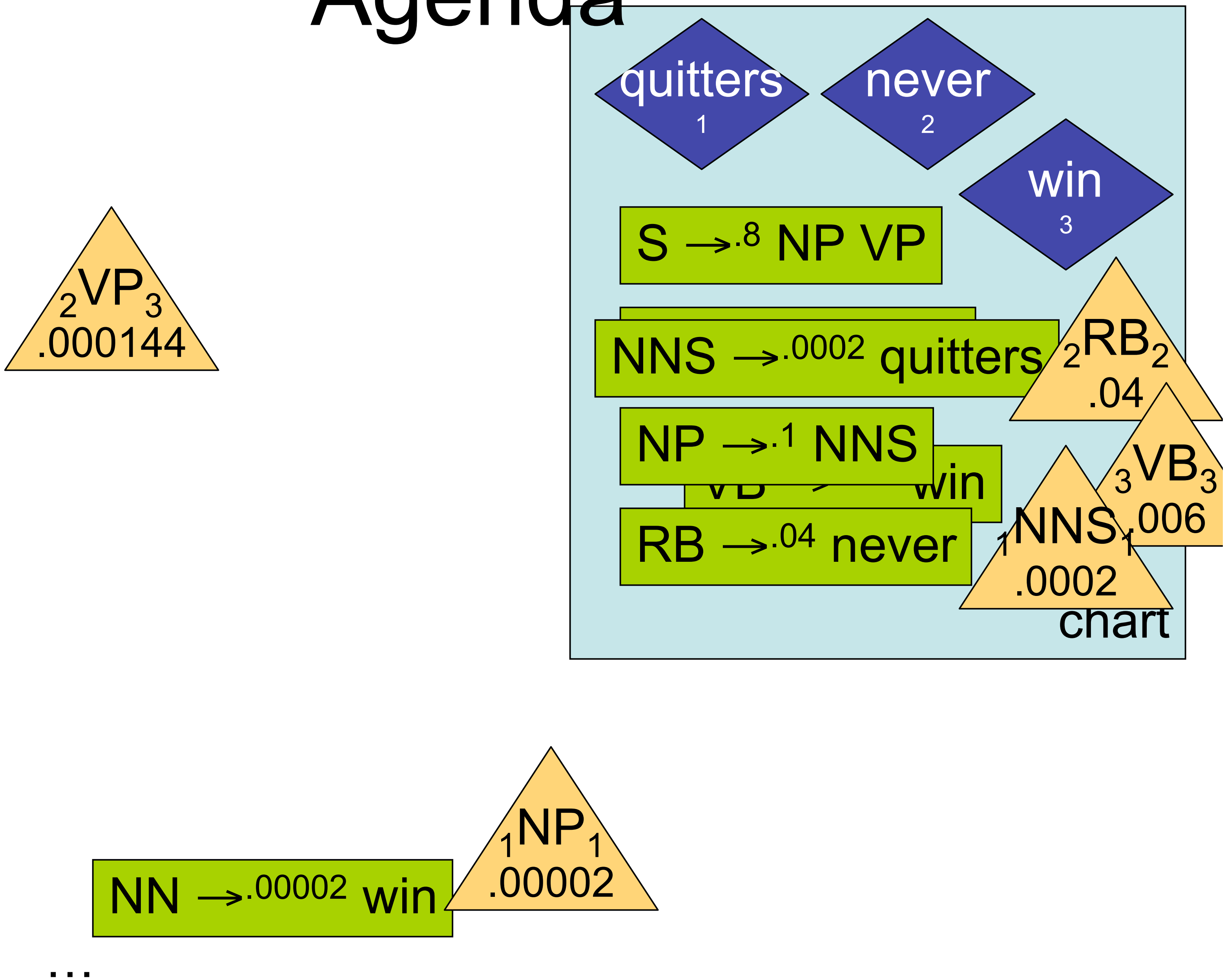


quitters
1
never
2
win
3
S →.8 NP VP
NNS →.0002 quitters
NP →.1 NNS
win
RB →.04 never
2RB2
.04
3VB3
.006
chart
1NNS1
.0002
1NP1
.00002
NN →.00002 win
2VP3
.000144
…

Agenda

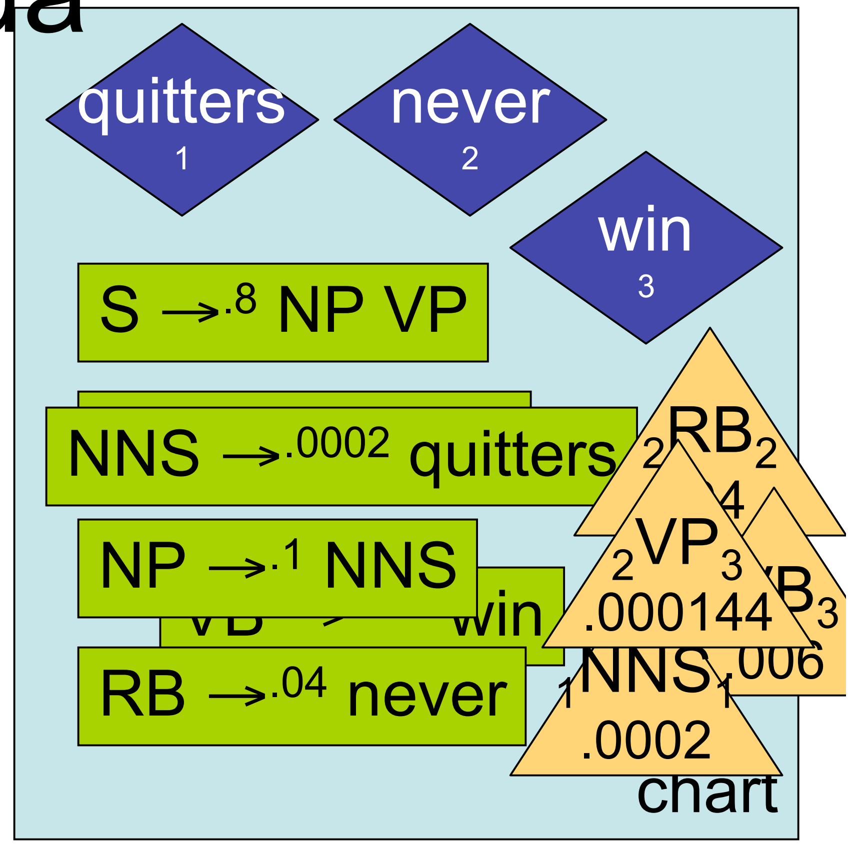

quitters
1
never
2
win
3
S →.8 NP VP
NNS →.0002 quitters
NP →.1 NNS
RB →.04 never
2RB2
.04
3VB3
.006
1NNS1
.0002
chart
NN →.00002 win
1NP1
.00002
2VP3
.000144
…

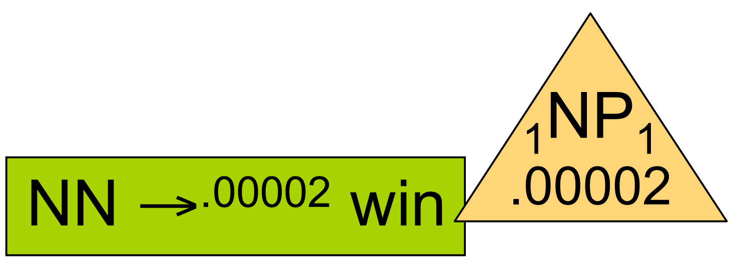
Agenda



quitters
1
never
2
win
3
S →.8 NP VP
NNS →.0002 quitters
NP →.1 NNS
win
RB →.04 never
2RB2
.04
3VB3
.006
1NNS1
.0002
chart
2VP3
.000144
NN →.00002 win
1NP1
.00002
…

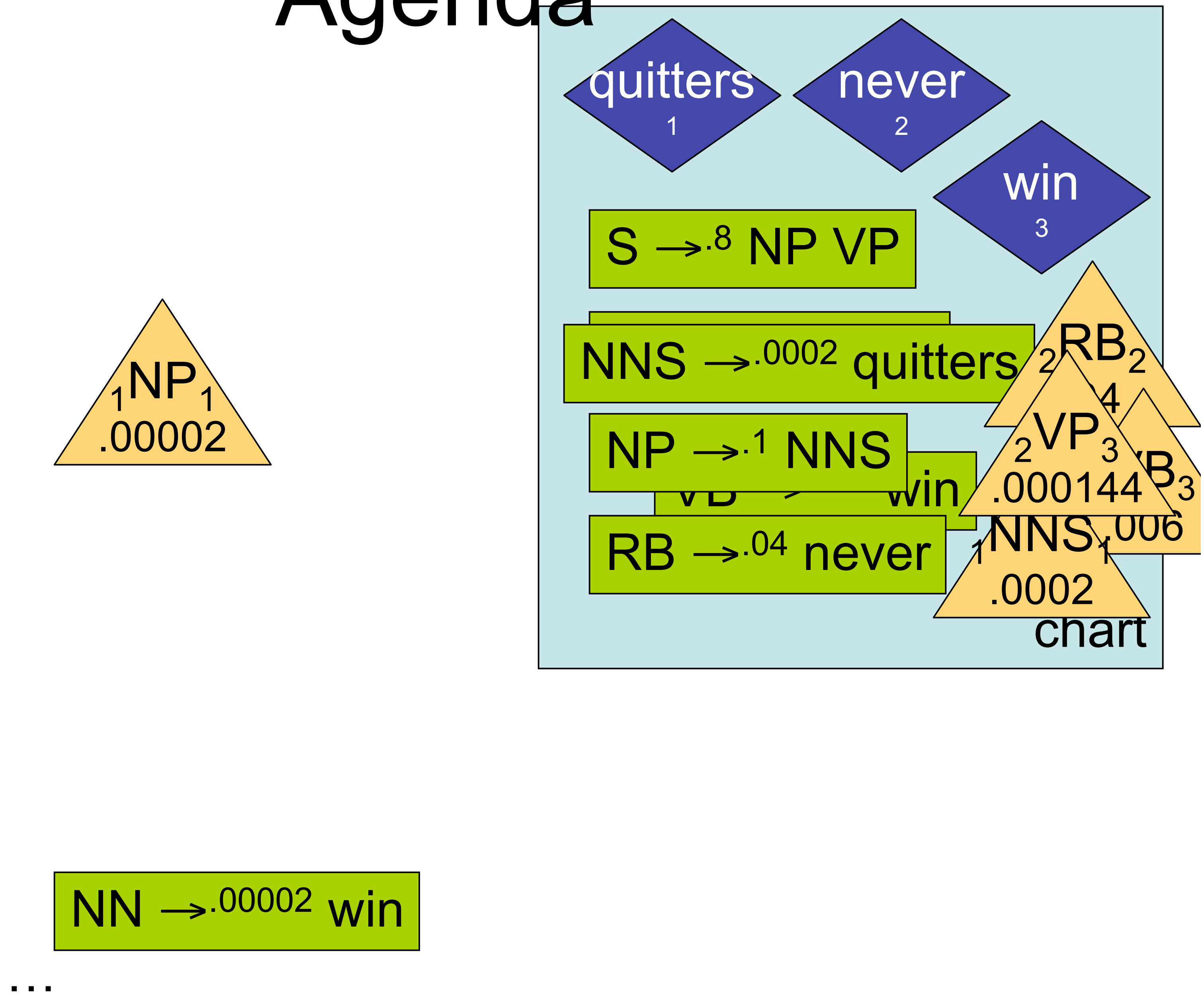
Agenda


quitters
1
never
2
win
3
S →.8 NP VP
NNS →.0002 quitters
NP →.1 NNS
RB →.04 never
2RB2
2VP3
.000144
.006
1NNS1
.0002
chart


NN →.00002 win
1NP1
.00002

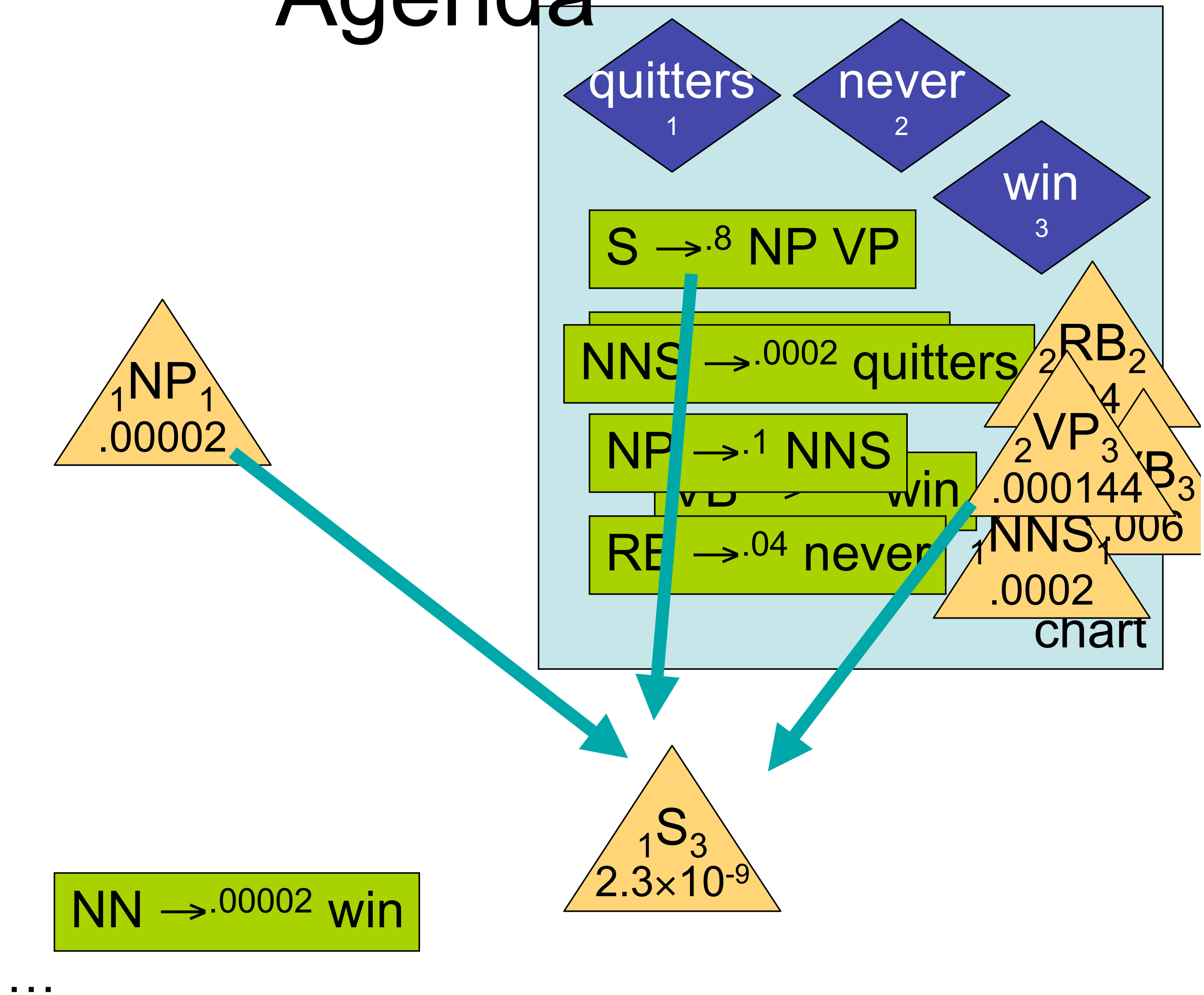
…

Agenda



quitters 1
never 2
win 3
S →.8 NP VP
NNS →.0002 quitters
NP →.1 NNS
RB →.04 never
2RB2
2VP3
.000144
.006
1NNS1
.0002
chart
1NP1
.00002
NN →.00002 win
…

Agenda



quitters
1
never
2
win
3
S →.8 NP VP
NNS →.0002 quitters
NP →.1 NNS
RB →.04 never
win
1NP1
.00002
2RB2
2VP3
.000144
.006
1NNS1
.0002
chart
1S3
2.3×10-9
NN →.00002 win
…

Best-First Parsing

- **Viterbi** semiring (find the best parse)
- Cf. Goodman, build the chart and fill in weights at the same time.
- Order items by their weights.
  - "**Uniform cost search**"
  - Guarantee: the first time **goal** is popped, you have the optimal parse.
- Charniak et al., 1998: heuristics to speed this up. "Figures of Merit" (big speed payoff).
- Klein and Manning (2003): admissible heuristics to guarantee optimal parse is found (big speed payoff).

Dyna

- **Dyna** is a high-level programming language (like Prolog) for weighted deduction.
- Source code looks like Prolog.
- Compiles into C++.
- Core algorithms:
  - Generalized weighted, **prioritized** agenda.
    - Allows the use of heuristics, including A*
  - Efficient “tape” mechanism for reverse computation.
    - Very similar to backpropagation.

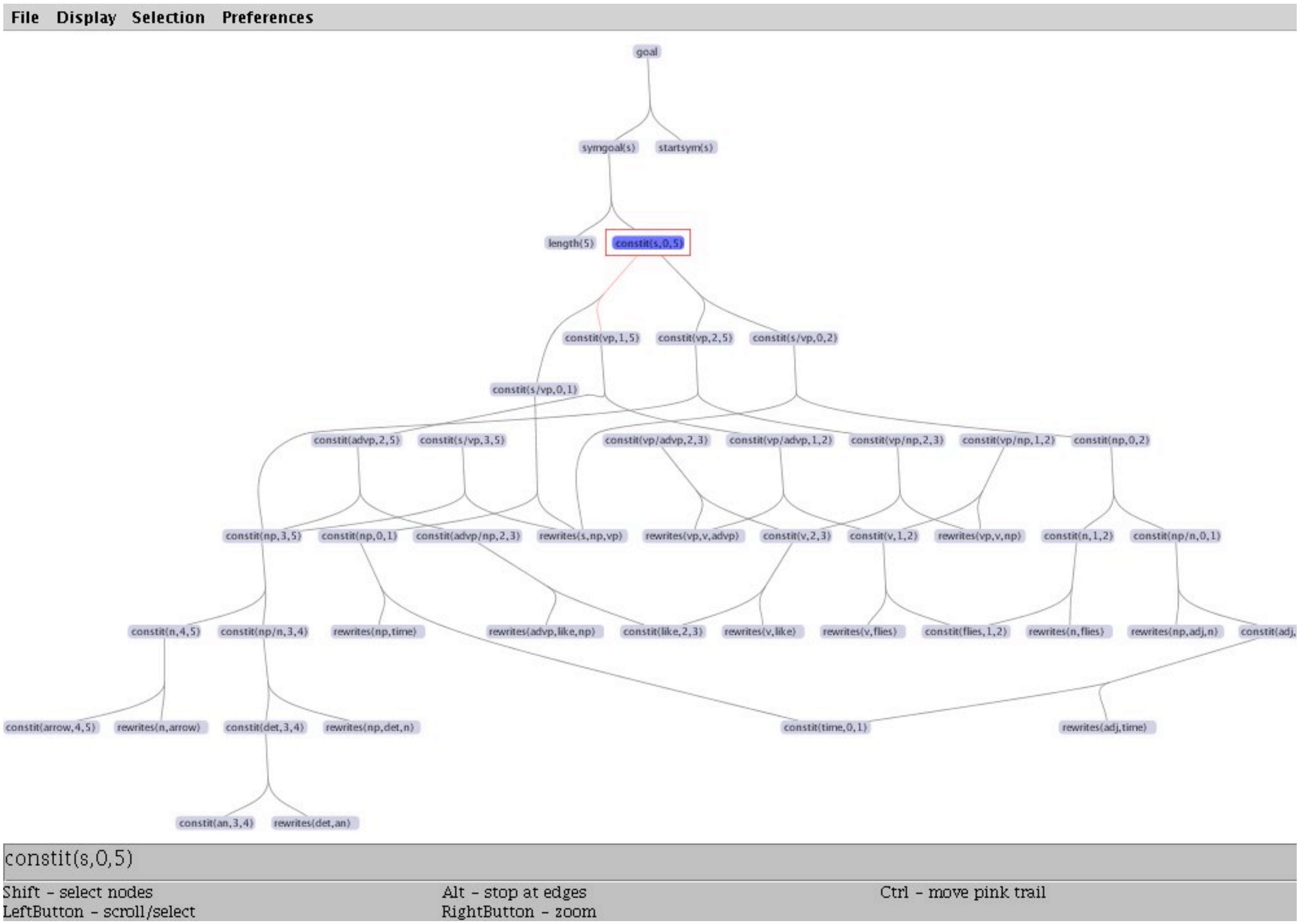
Dyna Programs

```
constit(X,I,J)    +=  word(W,I,J) * rewrite(X,W).
constit(X,I,J)    +=  constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal              +=  constit("s",0,N) whenever length(N).
```

```
constit(X,I,J)    max=  word(W,I,J) * rewrite(X,W).
constit(X,I,J)    max=  constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal              max=  constit("s",0,N) whenever length(N).
```

```
constit(X,I,J)    max=  word(W,I,J) * rewrite(X,W).
constit(X,I,J)    max=  constit(Y,I,Mid) * inter(X, Z, Mid, J).
inter(X, Z, Mid, J) max= constit(Z,Mid,J) * rewrite(X,Y,Z).
goal              max=  constit("s",0,N) whenever length(N).
```

Dyna Debugger



Parting Shots

- Weighted deduction as a convenient way to
  design,
  improve,
  understand,
  analyze,
  unify,
  and implement
  otherwise tricky dynamic programming algorithms.