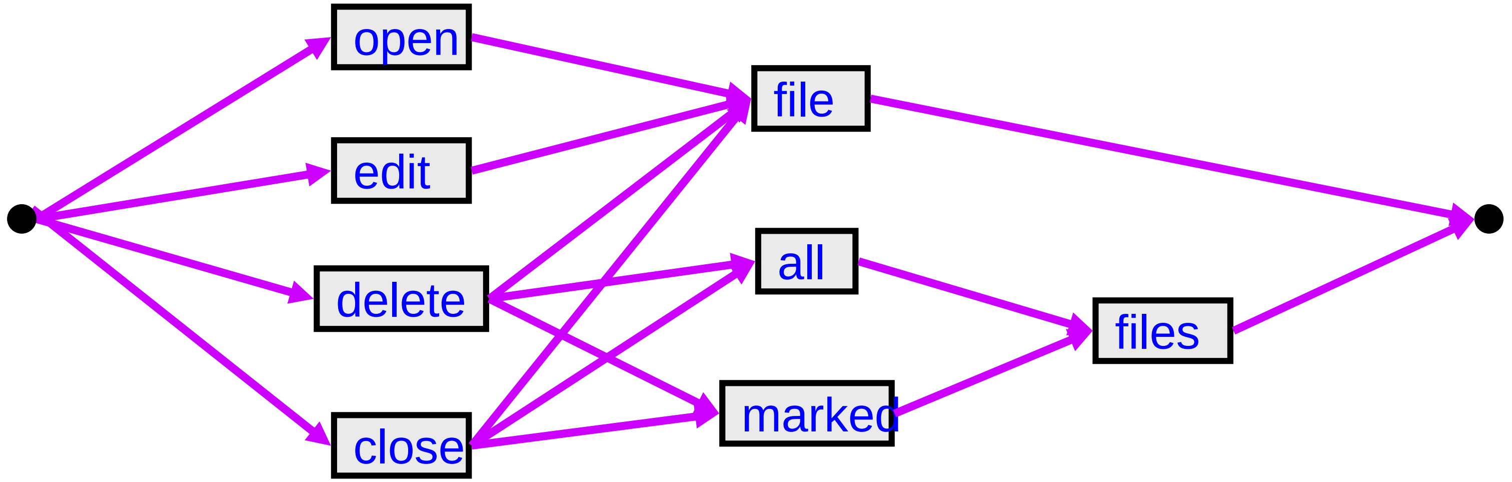
Design and Implementation of Speech Recognition Systems

Fall 2014
Ming Li

Class 12: Ngrams
Nov 11th

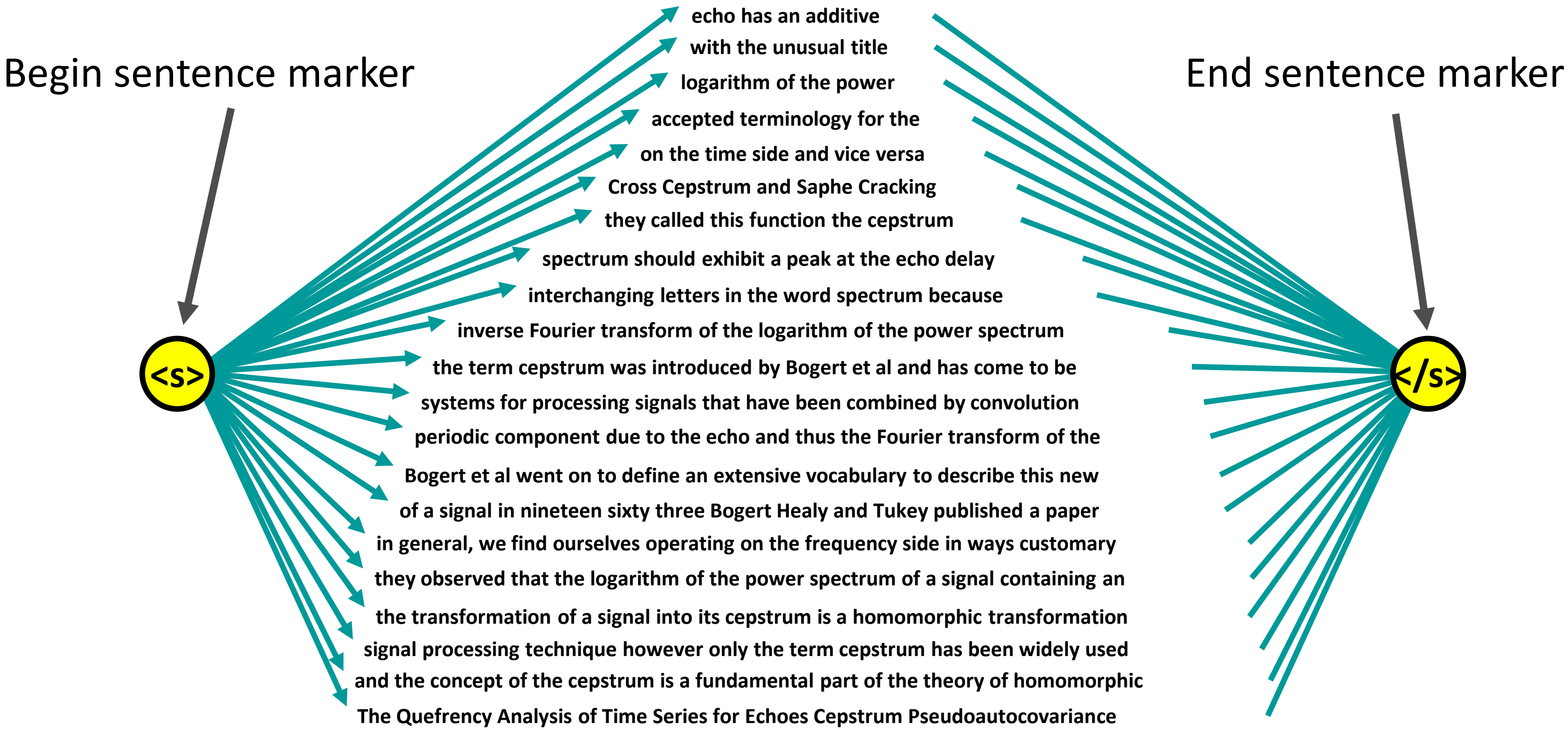
Thanks to Professor Bhiksha Raj for the contribution of the slides

Continuous speech recognition



- Compose a graph representing all possible word sequences
- Embed word HMMs in graph to form a “language” HMM
- Viterbi decode over the language HMM

What about free-form speech



- Graph is non-trivial
- Must express all sentences in the universe
  - With appropriate probabilities factored in
  - Can we simplify

The Bayes classifier for speech recognition

- The Bayes classification rule for speech recognition:

$$word_1, word_2, \ldots = \arg\max_{w_1, w_2, \ldots} P(X \mid w_1, w_2, ..)P(w_1, w_2, ..)$$

- $P(X \mid w_1, w_2, \ldots)$ = likelihood that speaking the word sequence w_1, w_2 … could result in the data (feature vector sequence) *X*
- $P(w_1, w_2 \ldots)$ measures the probability that a person might actually utter the word sequence w_1, w_2 ….
  - This will be 0 for impossible word sequences
- In theory, the probability term on the right hand side of the equation must be computed for every possible word sequence
- In practice this is often impossible
  - There are infinite word sequences

The Bayes classifier for speech recognition

Speech recognition system solves

$$word_1, word_2, \ldots, word_N =$$

$$\arg\max_{wd_1, wd_2, \ldots, wd_N} \{P(signal|wd_1, wd_2, \ldots, wd_N)P(wd_1, wd_2, \ldots, wd_N)\}$$

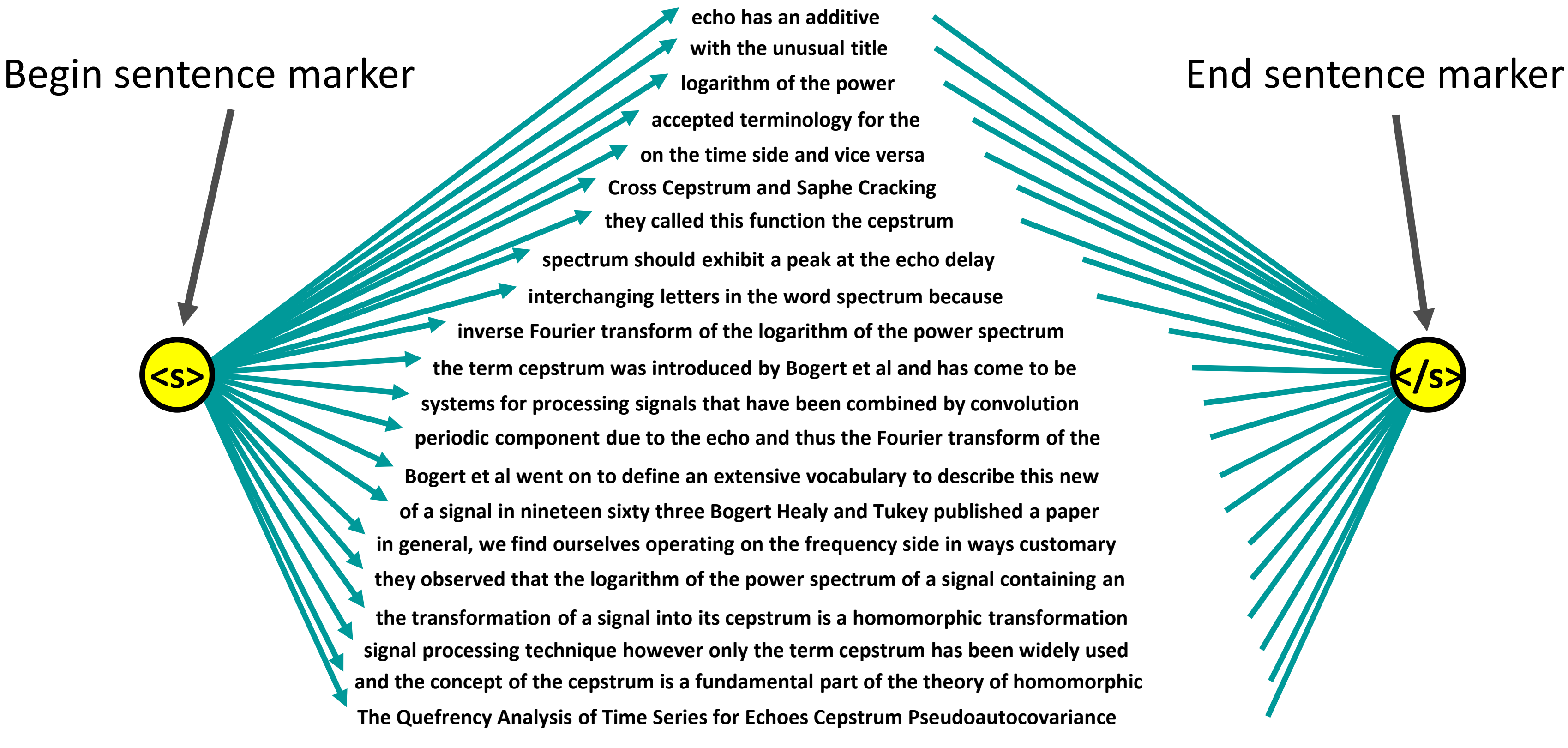
Acoustic model

For HMM-based systems this is an HMM

Lanugage model

$$\arg\max_{wd_1, wd_2, \ldots} \log P(signal \mid wd_1, wd_2, \ldots) + \log P(wd_1, wd_2, \ldots)$$

The complete language graph



- There will be one path for every possible word sequence
- *A priori* probability for a word sequence can be applied anywhere along the path representing that word sequence.
- It is the structure and size of this graph that determines the feasibility of the recognition task

A left-to-right model for the langauge

- A factored representation of *a priori* probability of a word sequence

  P(<s> word1 word2 word3 word4…</s>) =
  P(<s>) P(word1 | <s>) P(word2 | <s> word1) P(word3 | <s> word1 word2)…

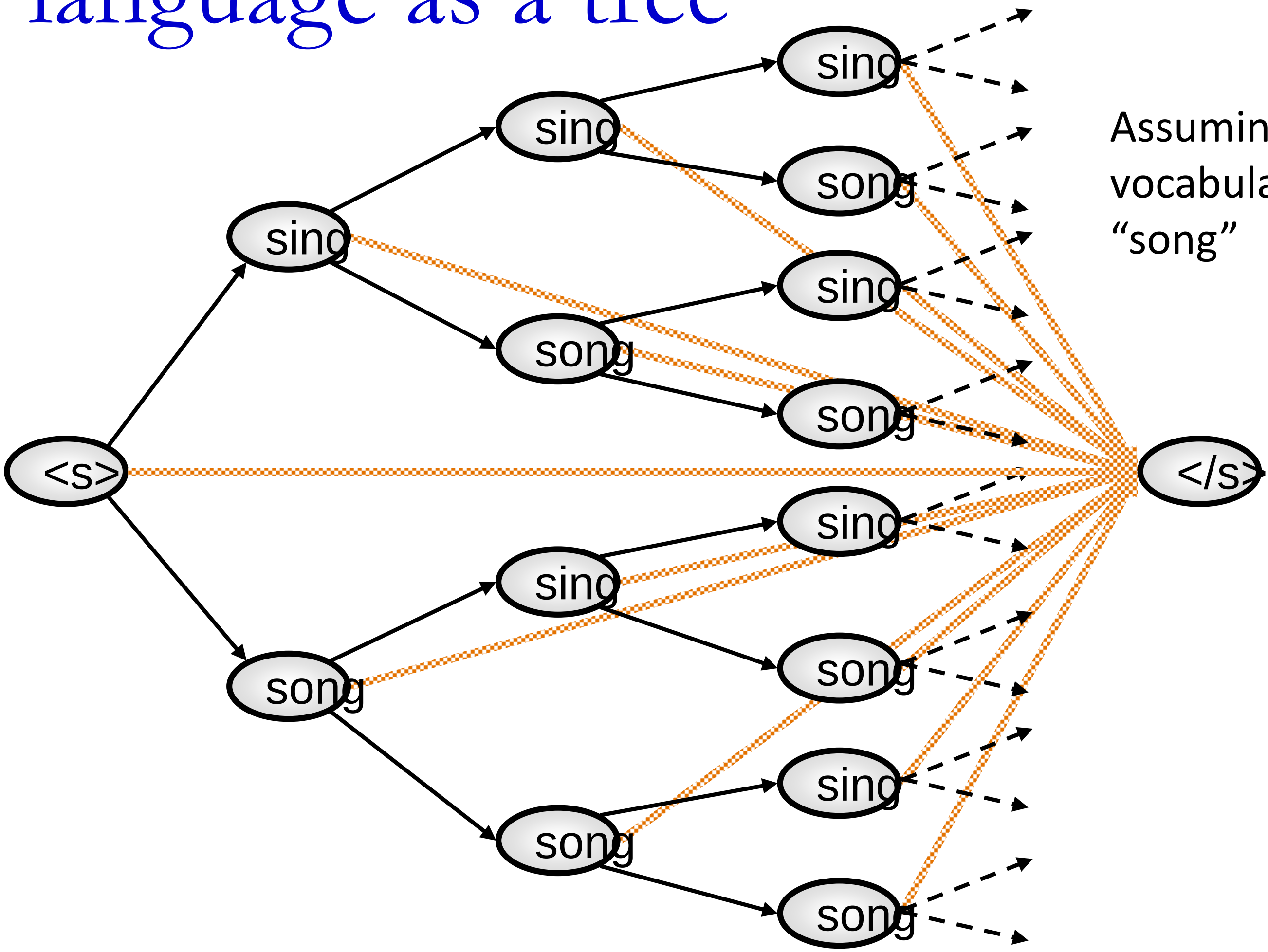
- This is a left-to-right factorization of the probability
  - The probability of a word assumed dependent only on the words preceding it
  - This probability model for word sequences is as accurate as the earlier whole-word-sequence model, in theory

- It has the advantage that the probabilities of words are incrementally applied
  - This is perfect for speech recognition

- P(word1 word2 word3 word4 …) is incrementally obtained :

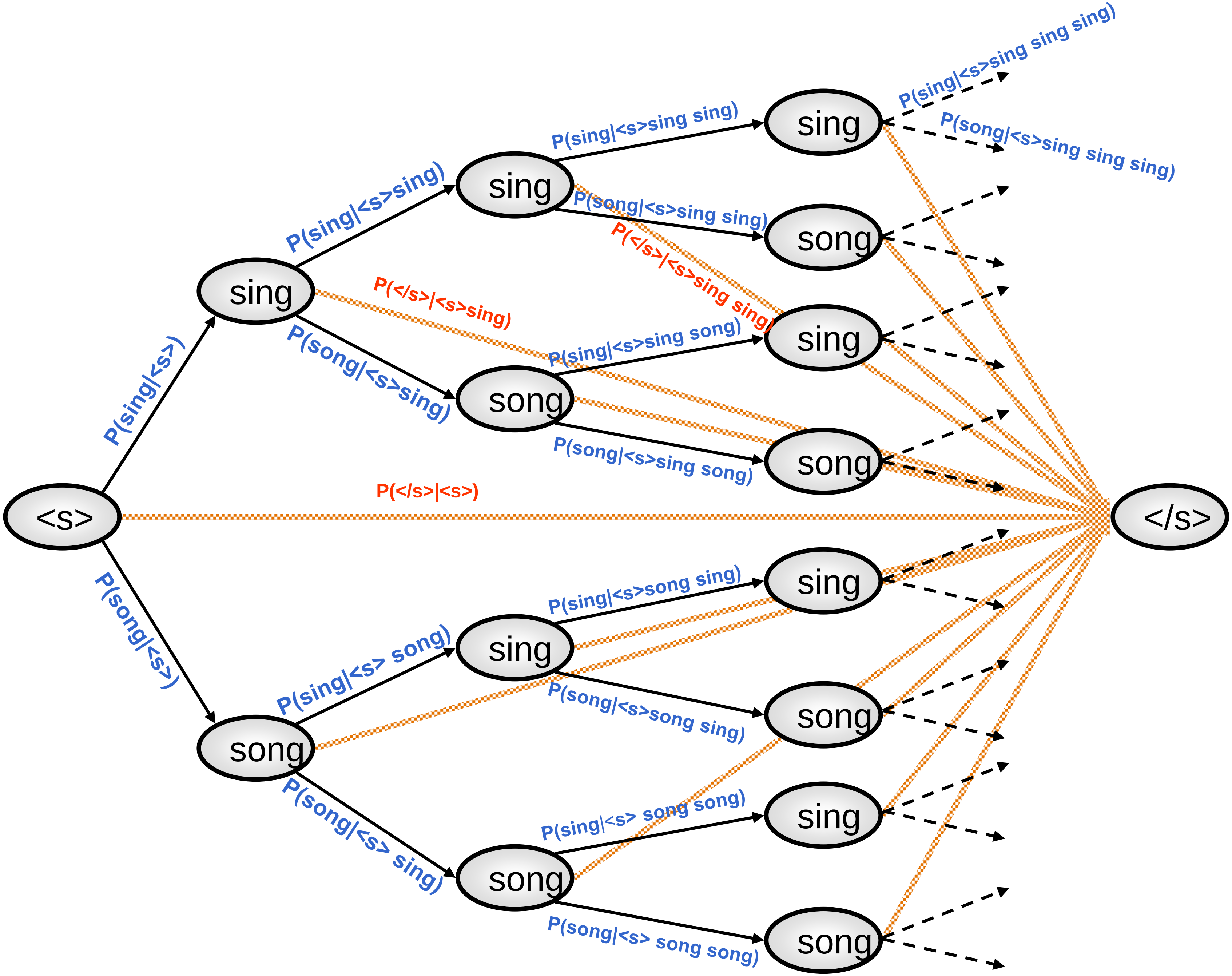
  word1
  word1 **word2**
  word1 word2 **word3**
  word1 word2 word3 **word4**
  …..

The language as a tree



Assuming a two-word vocabulary: “sing” and “song”

- *A priori* probabilities for word sequences are spread through the graph
  - They are applied on every edge
- This is a much more compact representation of the language than the full graph shown earlier
  - But is still infinitely large in size


<s>
sing
song
sing
song
sing
song
sing
song
sing
song
sing
song
sing
song
</s>
P(sing|<s>)
P(song|<s>)
P(</s>|<s>)
P(sing|<s>sing)
P(</s>|<s>sing)
P(song|<s>sing)
P(sing|<s> song)
P(song|<s> sing)
P(sing|<s>sing sing)
P(song|<s>sing sing)
P(</s>|<s>sing sing)
P(sing|<s>sing song)
P(song|<s>sing song)
P(sing|<s>song sing)
P(song|<s>song sing)
P(sing|<s> song song)
P(song|<s> song song)
P(sing|<s>sing sing sing)
P(song|<s>sing sing sing)

The N-gram model

- The N-gram assumption

$$P(w_K \mid w_1, w_2, w_3, \ldots w_{K-1}) = P(w_K \mid w_{K-(N-1)}, w_{K-(N-2)}, \ldots, w_{K-1})$$

- The probability of a word is assumed to be dependent only on the past N-1 words
  - For a 4-gram model, the probability that "two times two is" is followed by "four" is assumed identical to the probability "seven times two is" is followed by "four".
- This is not such a poor assumption
  - Surprisingly, the words we speak (or write) at any time are largely dependent on the previous 3-4 words.

The validity of the N-gram assumption

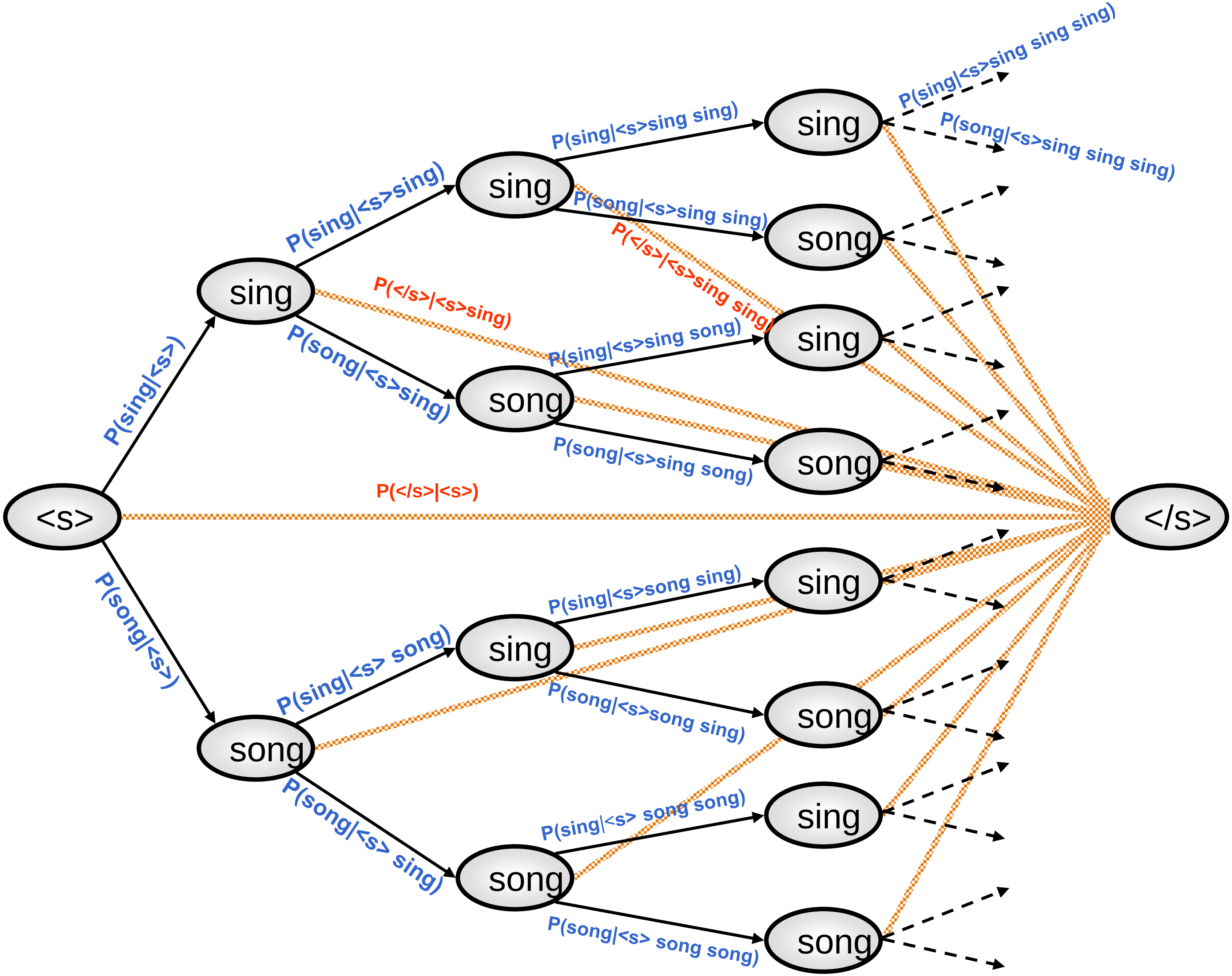
- An N-gram language model is a generative model
  - One can generate word sequences randomly from it
- In a good generative model, randomly generated word sequences should be similar to word sequences that occur naturally in the language
  - Word sequences that are more common in the language should be generated more frequently
- Is an N-gram language model a good model?
  - Does it generate reasonable sentences
- Thought exercise: how would you generate word sequences from an N-gram LM ?
  - Clue: N-gram LMs include the probability of a sentence end marker

N-gram LMs

- N-gram models are reasonably good models for the language at higher N
  - As N increases, they become better models
- For lower N (N=1, N=2), they are not so good as generative models
- Nevertheless, they are quite effective for *analyzing* the relative validity of word sequences
  - Which of a given set of word sequences is more likely to be valid
  - They usually assign higher probabilities to plausible word sequences than to implausible ones
- This, and the fact that they are left-to-right (Markov) models makes them very popular in speech recognition
  - They have found to be the most effective language models for large vocabulary speech recognition

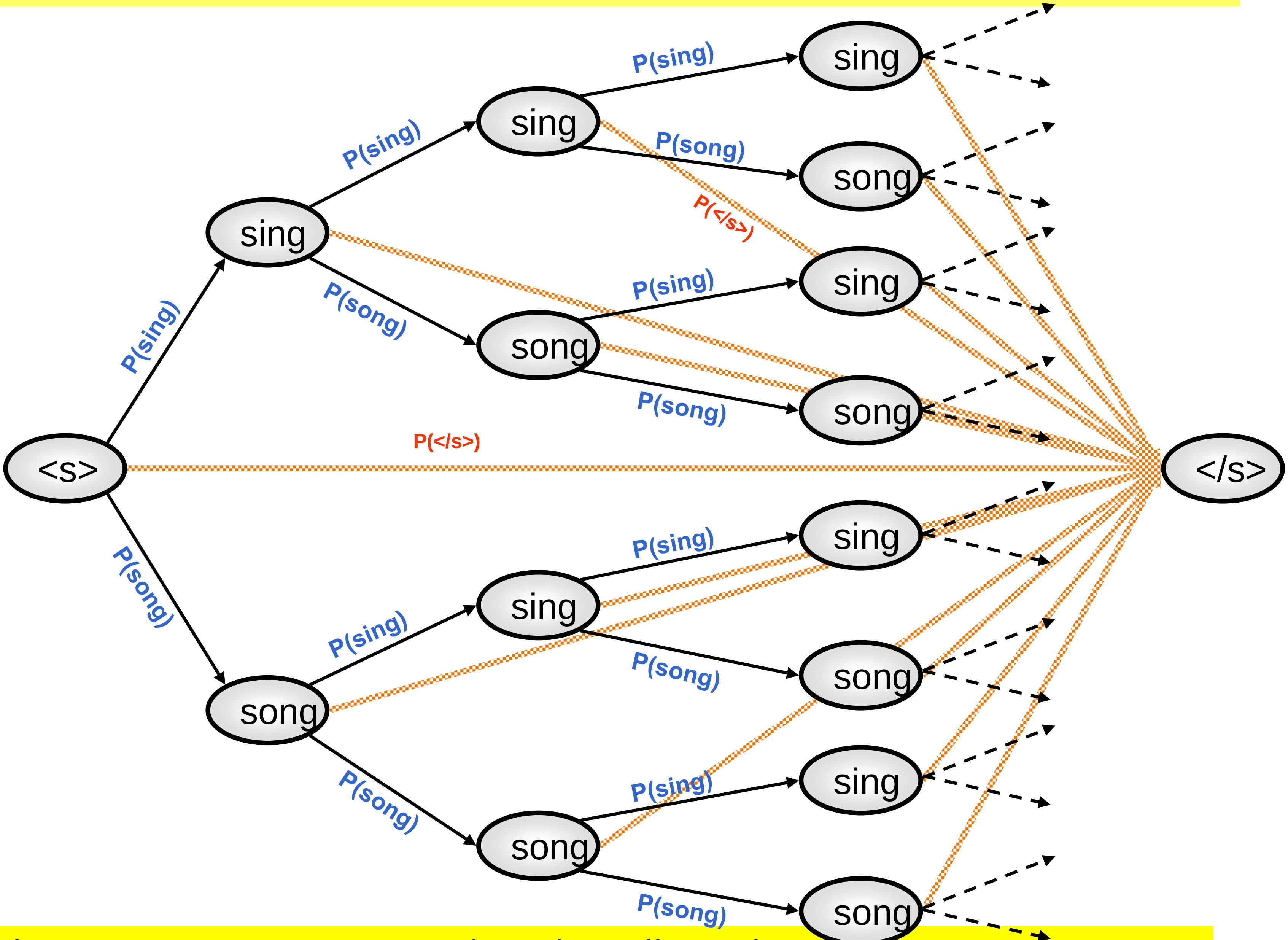
N-gram LMs and compact graphs

- By restricting the order of the N-gram LM, the infinite tree for the language can be collapsed into finite-sized graphs.
- Best explained with an example
- Consider the simple 2-word example with the words "Sing" and "Song" seen earlier
  - Word sequences are
    - Sing
    - Sing sing
    - Sing song sing
    - Sing sing song
    - Song
    - Song sing sing sing sing sing song
    - ….
  - There are infinite possible sequences

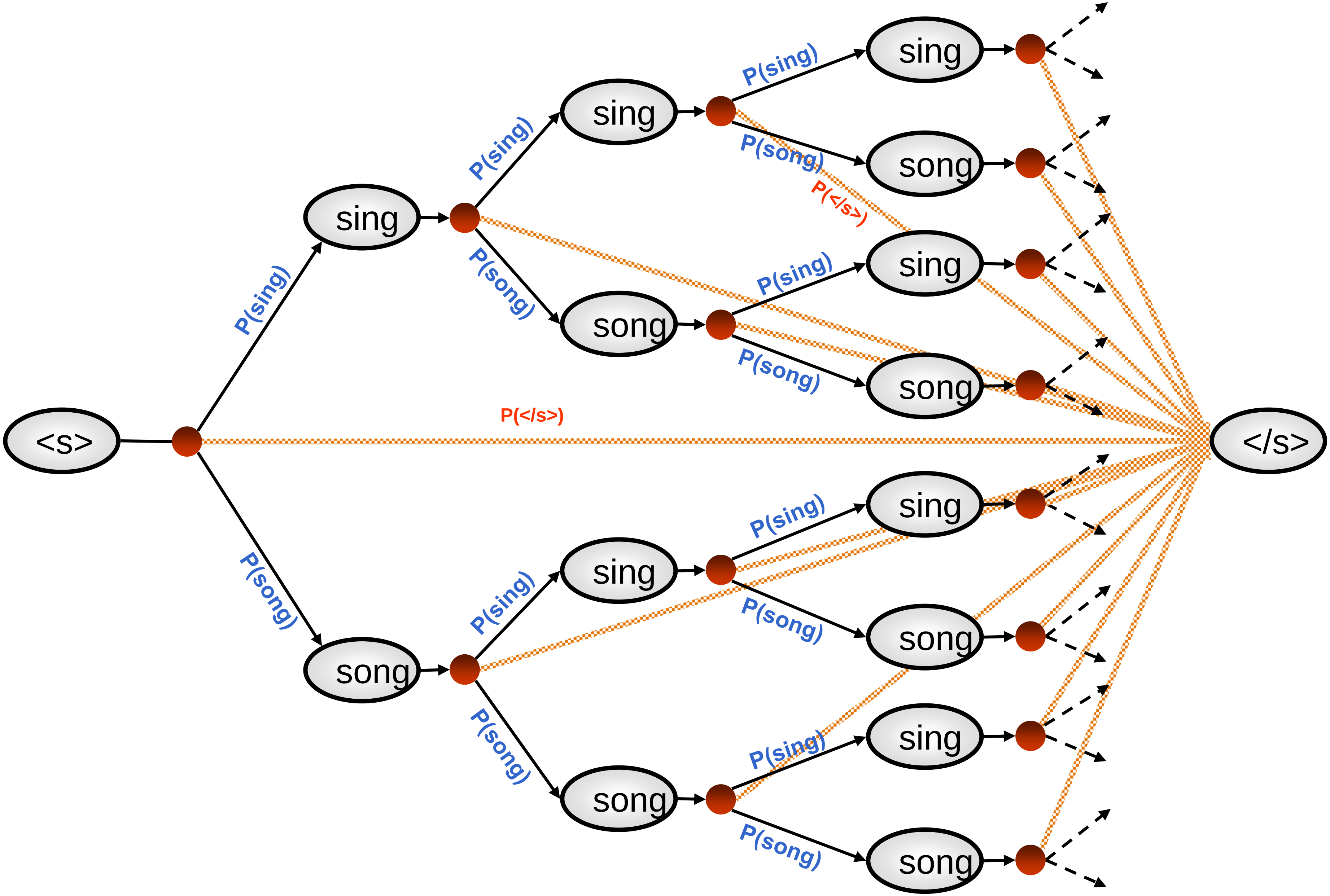


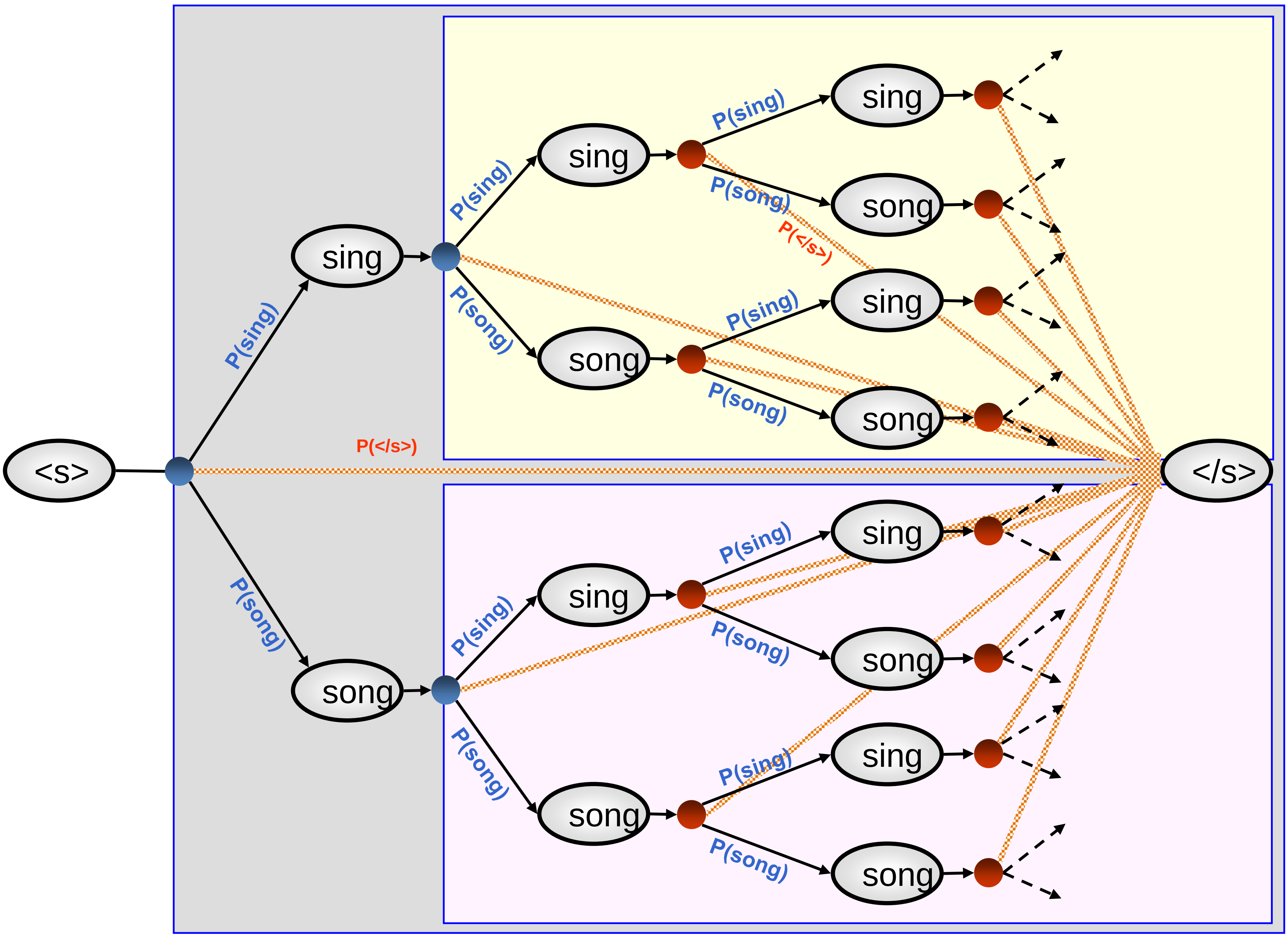
<s>
</s>
sing
song
P(sing|<s>)
P(song|<s>)
P(</s>|<s>)
P(sing|<s>sing)
P(song|<s>sing)
P(</s>|<s>sing)
P(sing|<s> song)
P(song|<s> sing)
P(sing|<s>sing sing)
P(song|<s>sing sing)
P(</s>|<s>sing sing)
P(sing|<s>sing song)
P(song|<s>sing song)
P(sing|<s>song sing)
P(song|<s>song sing)
P(sing|<s> song song)
P(song|<s> song song)
P(sing|<s>sing sing sing)
P(song|<s>sing sing sing)

The two-word example as a full tree with a unigram LM

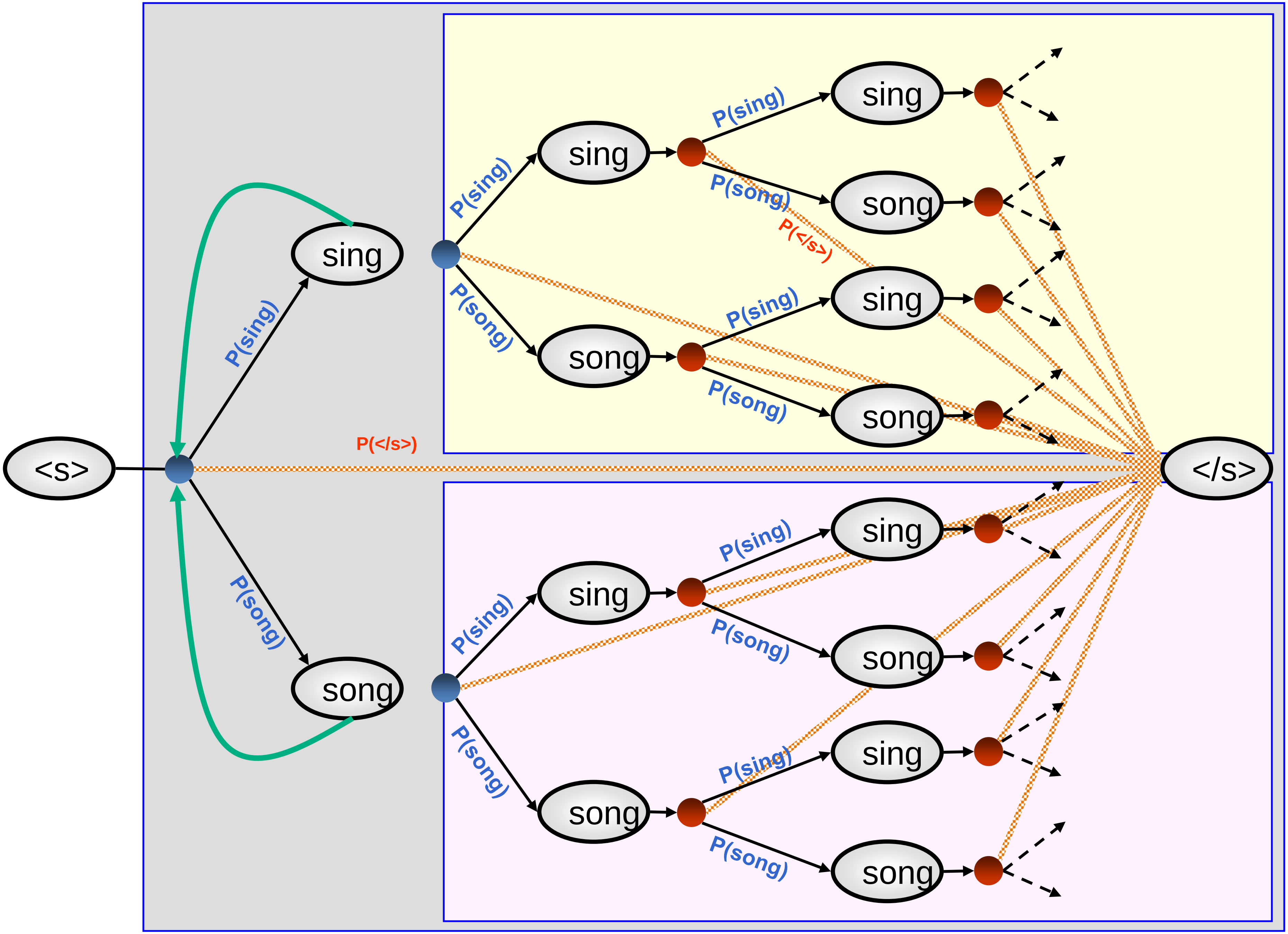


- The structure is recursive and can be collapsed


<s>
P(sing)
P(song)
P(</s>)
sing
song
P(sing)
P(song)
P(sing)
P(song)
sing
song
sing
song
P(sing)
P(song)
P(sing)
P(song)
P(sing)
P(song)
P(sing)
P(song)
P(</s>)
sing
song
sing
song
sing
song
sing
song
</s>

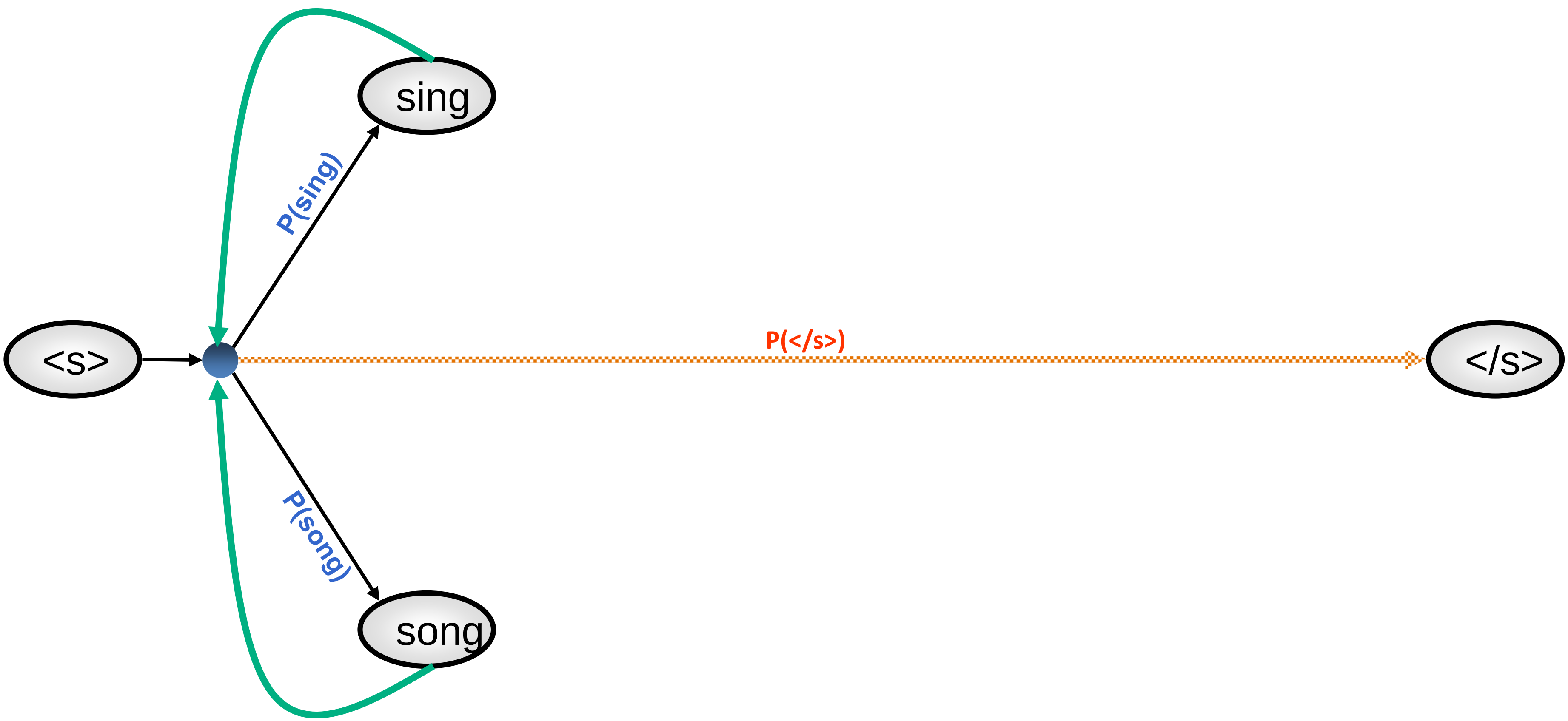


<s>
sing
song
sing
song
sing
song
sing
song
sing
song
sing
song
sing
song
P(sing)
P(song)
P(sing)
P(song)
P(sing)
P(song)
P(sing)
P(song)
P(sing)
P(song)
P(sing)
P(song)
P(sing)
P(song)
P(</s>)
P(</s>)
</s>

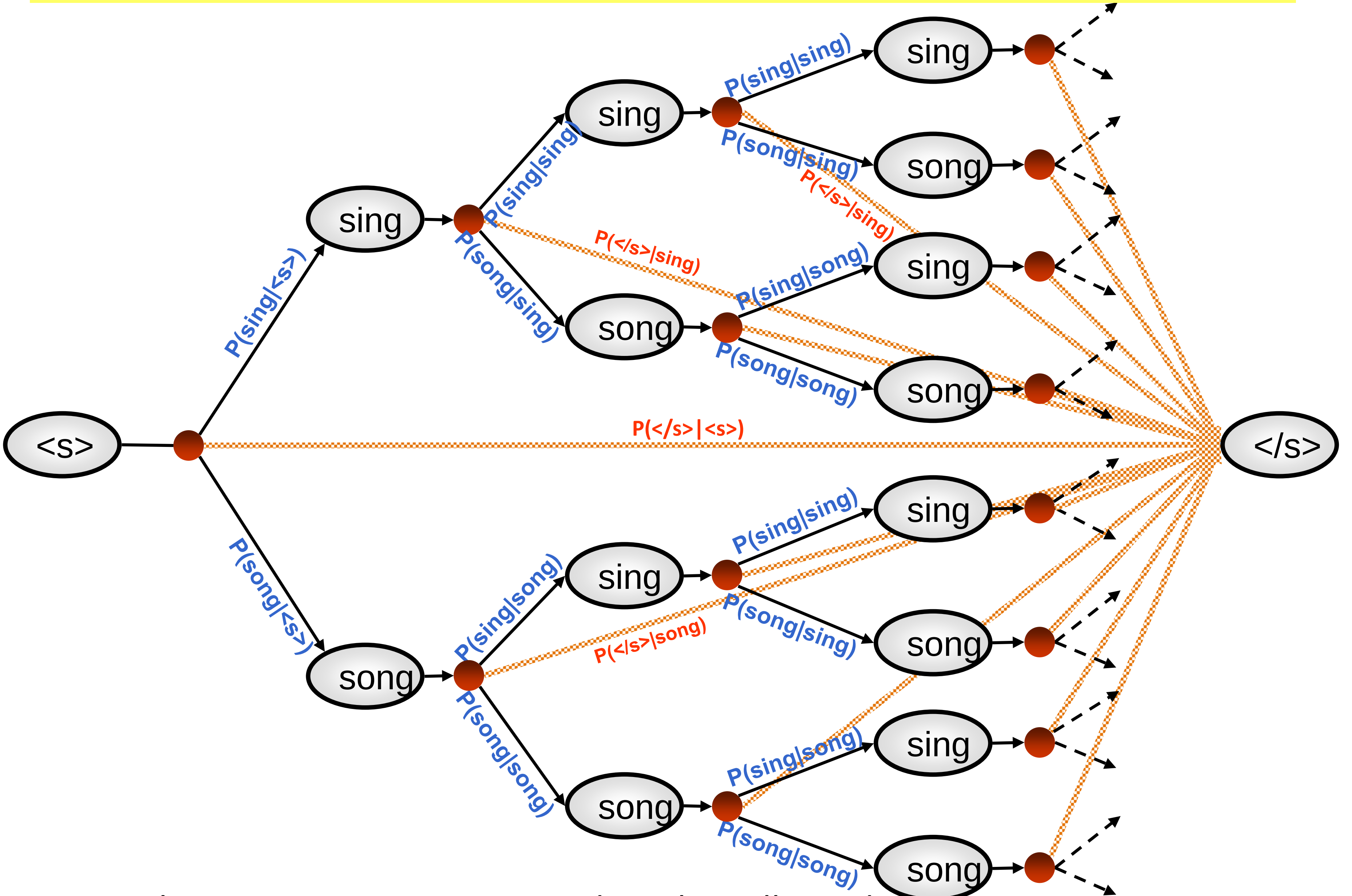


<s>
sing
song
P(sing)
P(song)
P(</s>)
</s>

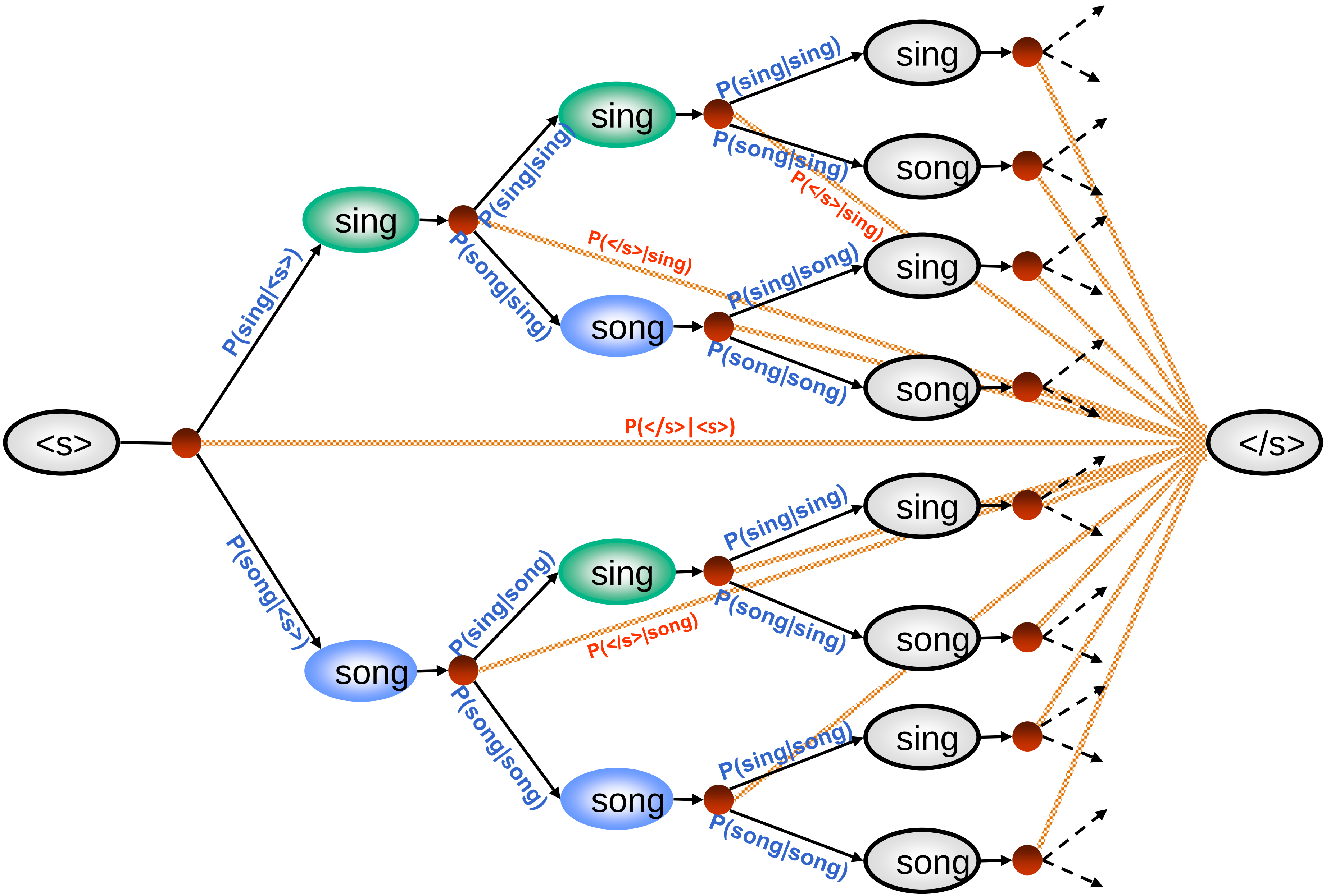
The two-word example with a unigram LM

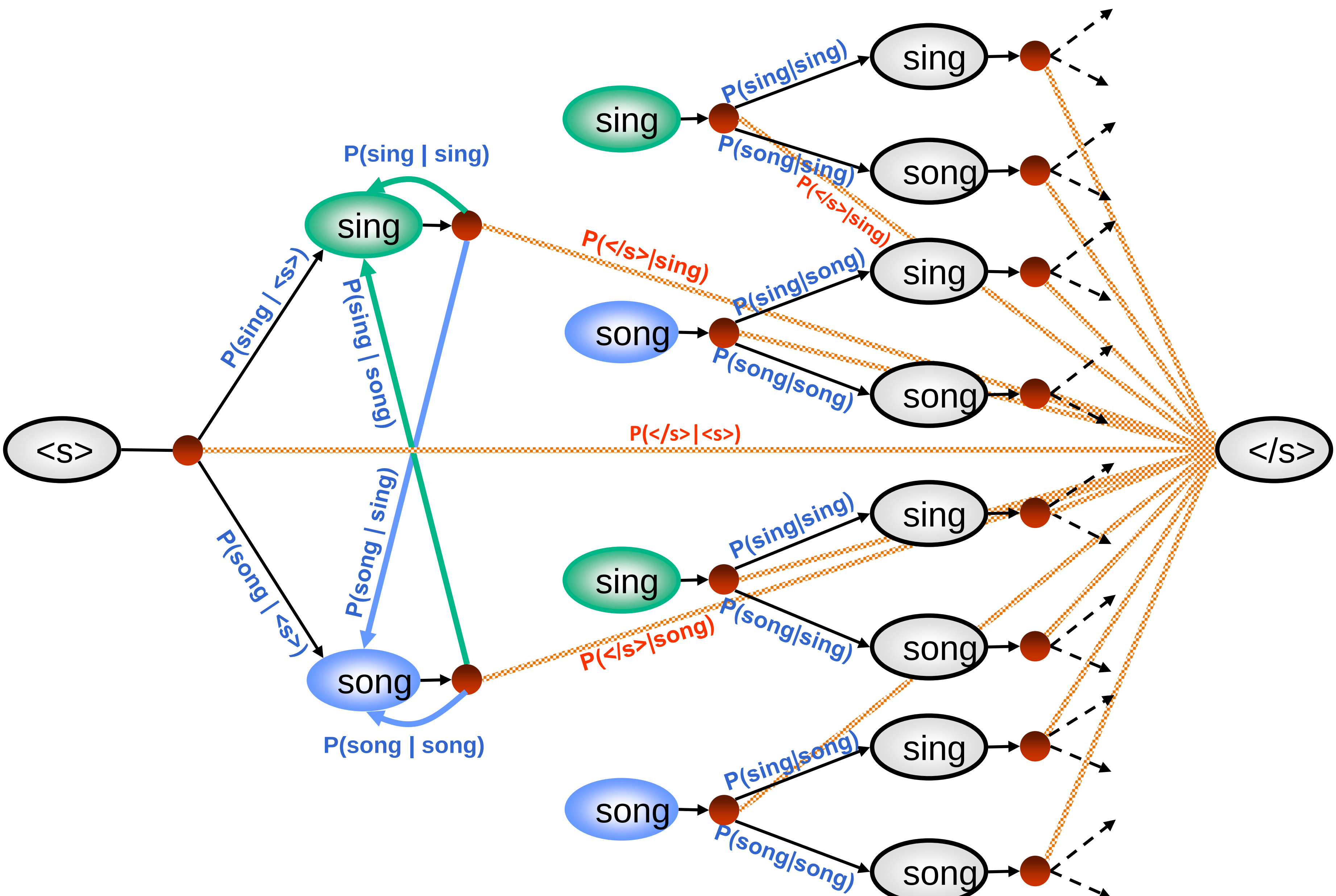


The two-word example as a full tree with a bigram LM



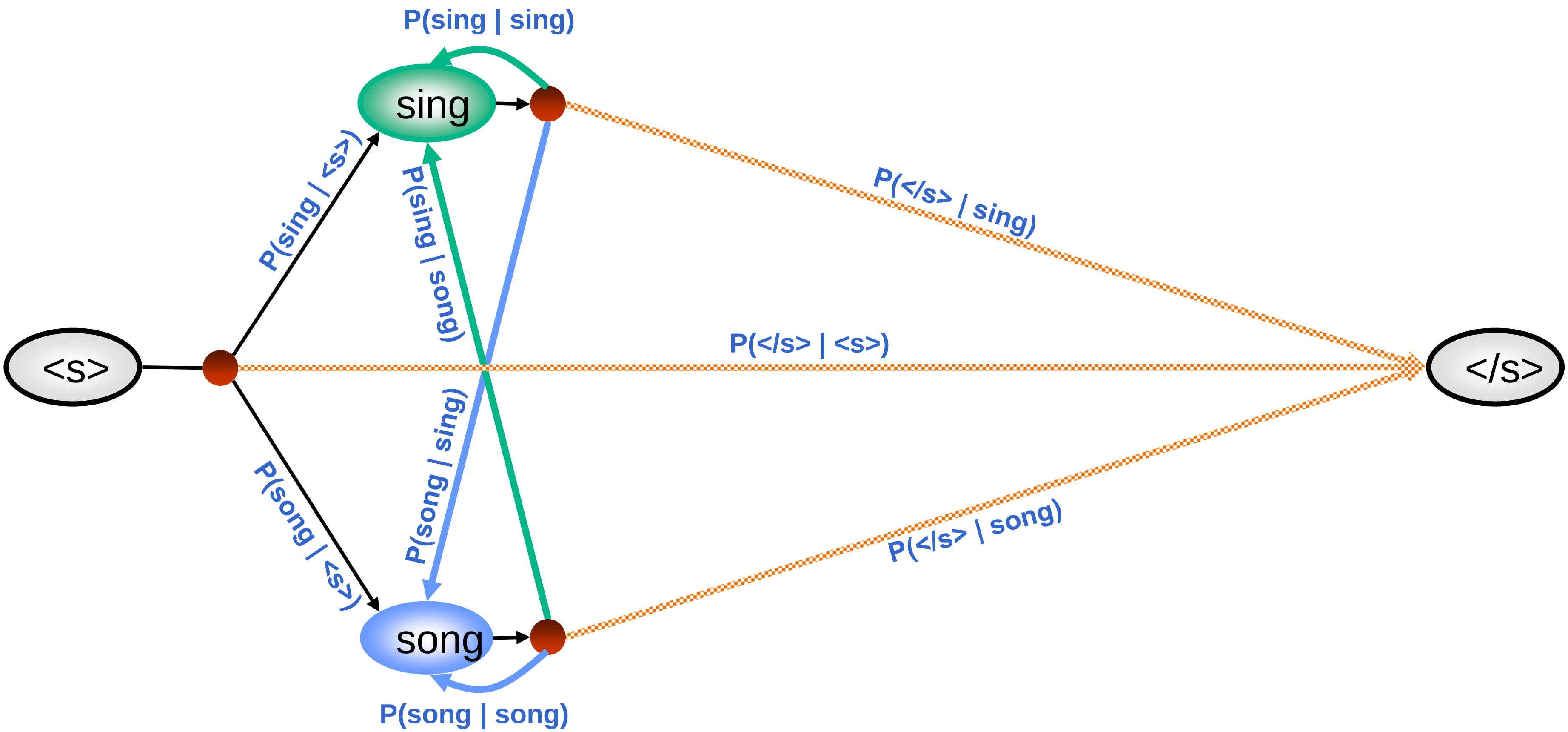
- The structure is recursive and can be collapsed


<s>
P(sing|<s>)
P(song|<s>)
P(</s>|<s>)
sing
song
P(sing|sing)
P(song|sing)
P(</s>|sing)
sing
song
P(sing|song)
P(song|song)
P(</s>|song)
sing
P(sing|sing)
P(song|sing)
P(</s>|sing)
song
P(sing|song)
P(song|song)
sing
song
sing
song
sing
P(sing|sing)
P(song|sing)
song
P(sing|song)
P(song|song)
sing
song
sing
song
</s>

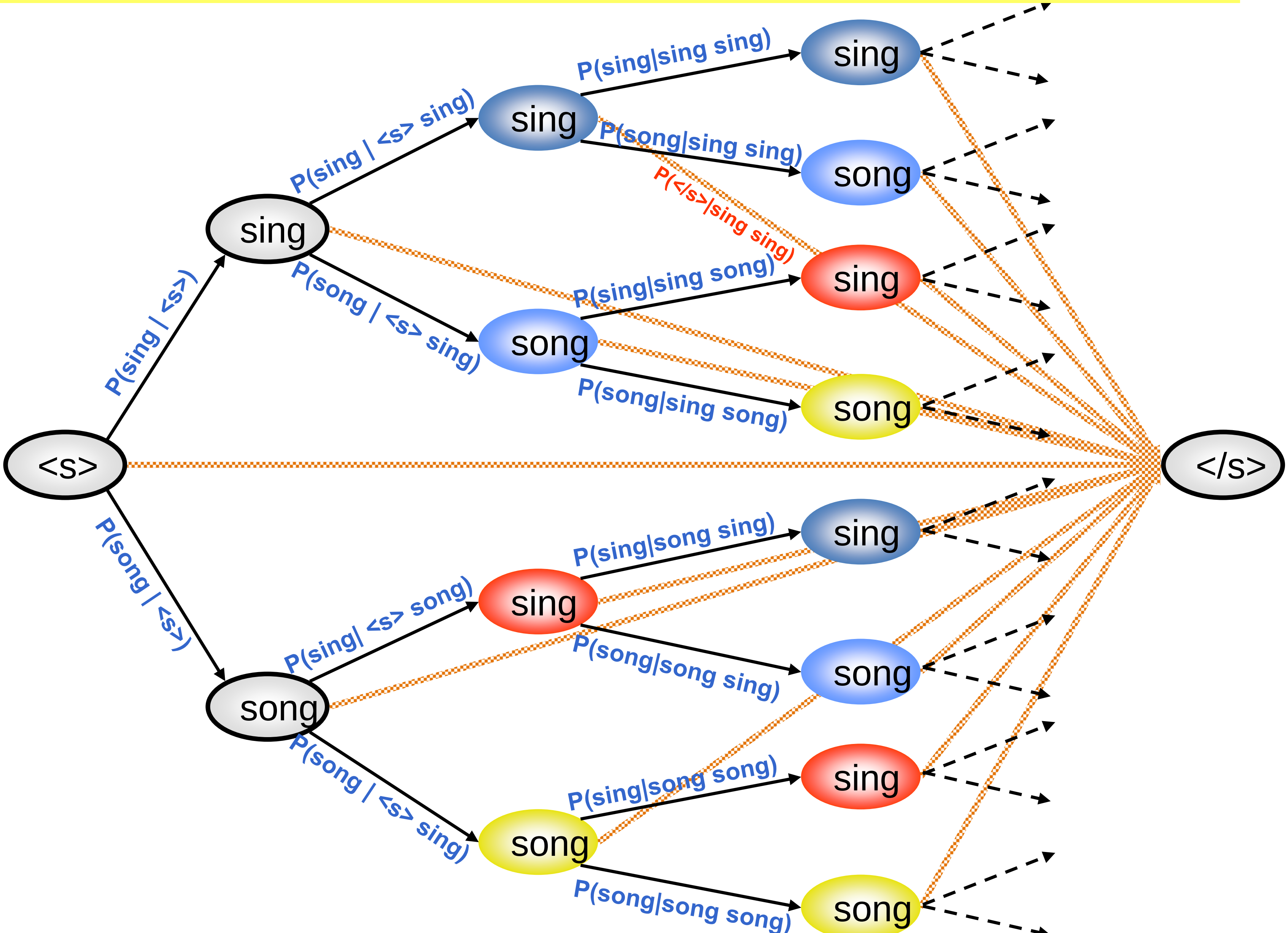


<s>
P(sing | <s>)
P(song | <s>)
sing
song
P(sing | sing)
P(song | song)
P(sing | song)
P(song | sing)
P(</s>|sing)
P(</s>|song)
P(</s>|<s>)
P(sing|sing)
P(song|sing)
P(</s>|sing)
P(sing|song)
P(song|song)
</s>

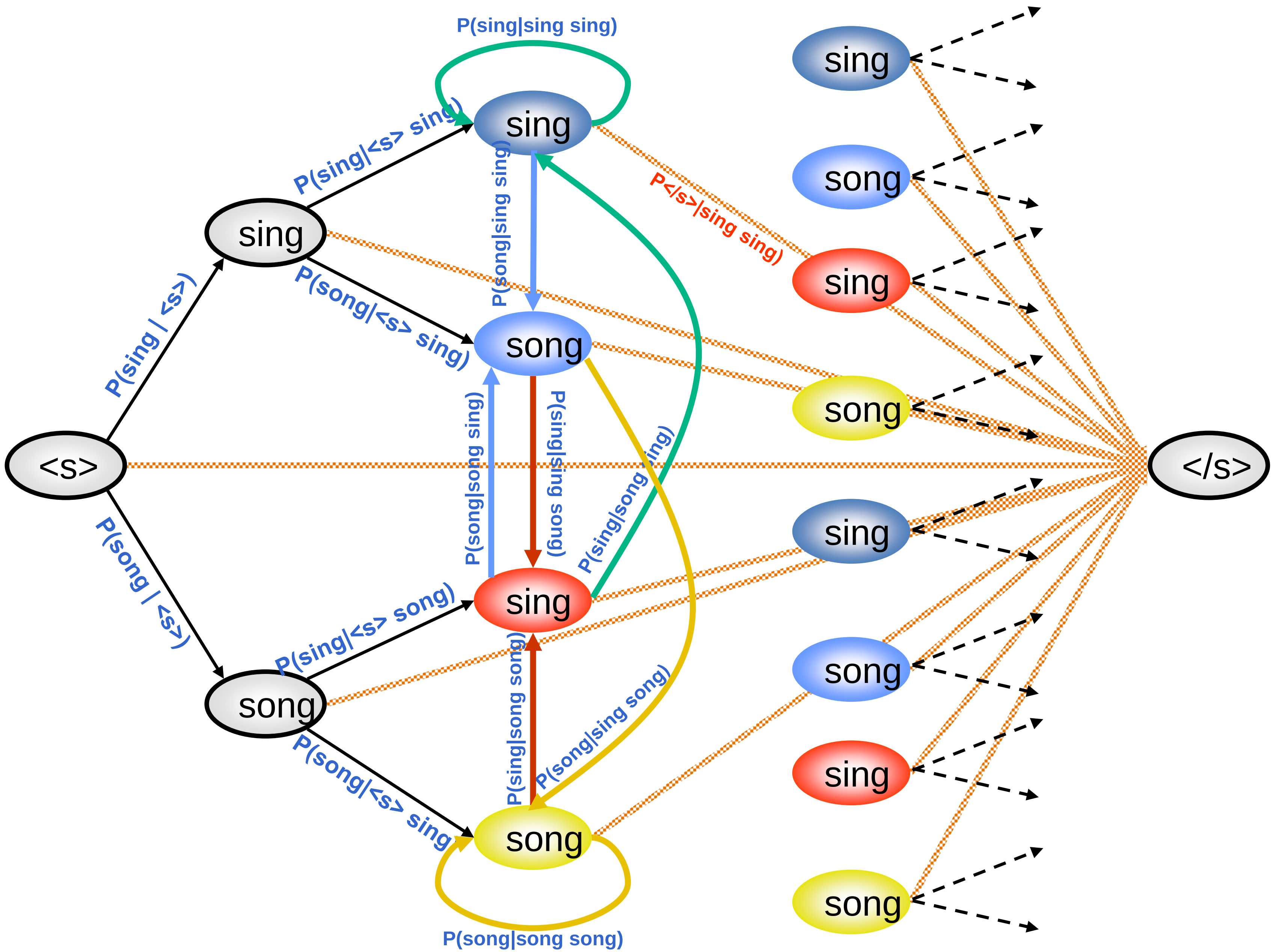
The two-word example with a bigram LM



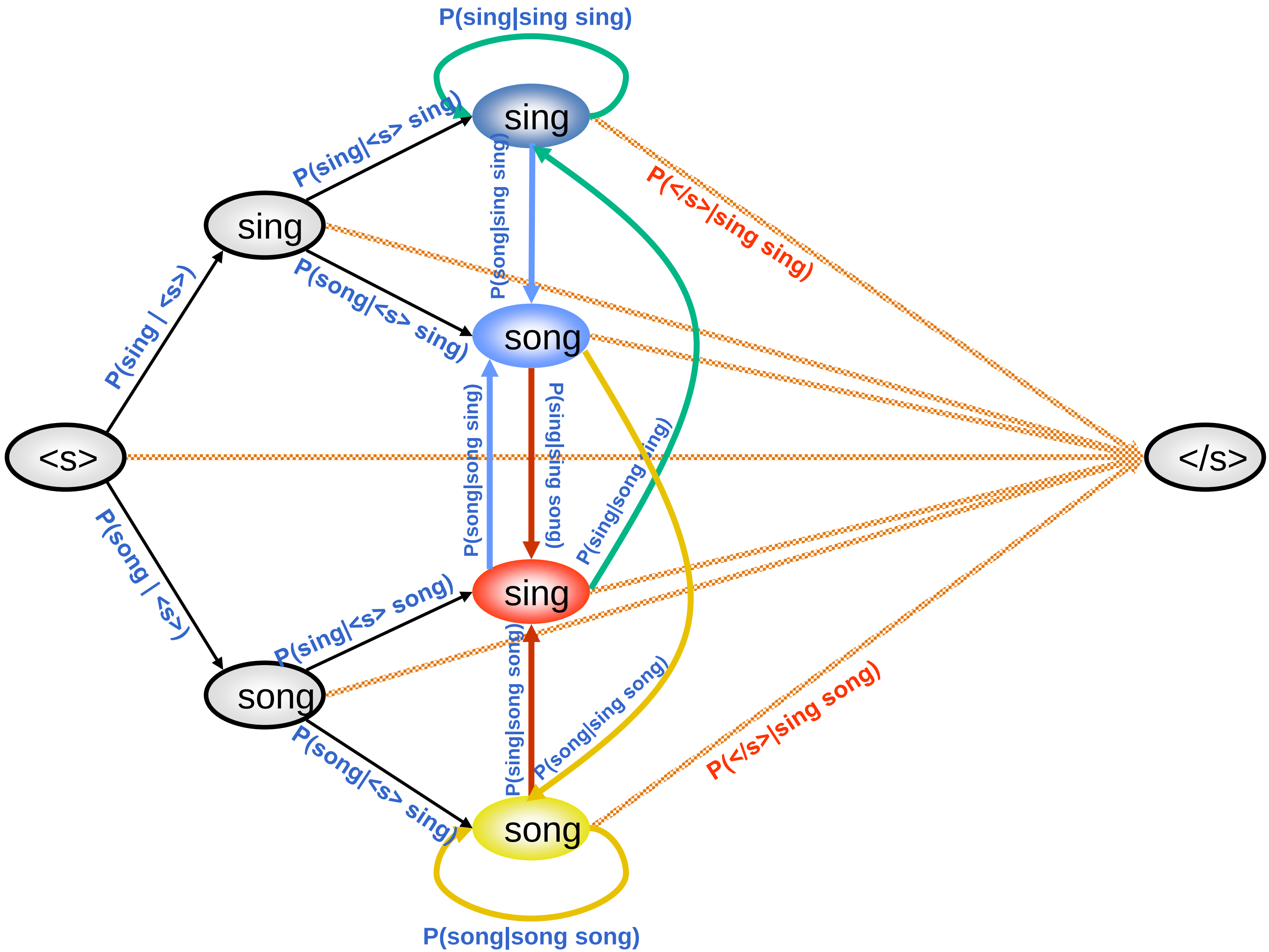
The two-word example as a full tree with a trigram LM



- The structure is recursive and can be collapsed

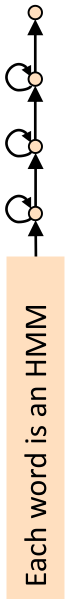


<s>
P(sing | <s>)
P(song | <s>)
sing
song
P(sing|<s> sing)
P(song|<s> sing)
P(sing|<s> song)
P(song|<s> sing)
P(sing|sing sing)
P(song|sing sing)
P(song|song sing)
P(sing|sing song)
P(sing|song sing)
P(sing|song song)
P(song|sing song)
P(song|song song)
P</s>|sing sing)
sing
song
sing
song
sing
song
sing
song
</s>



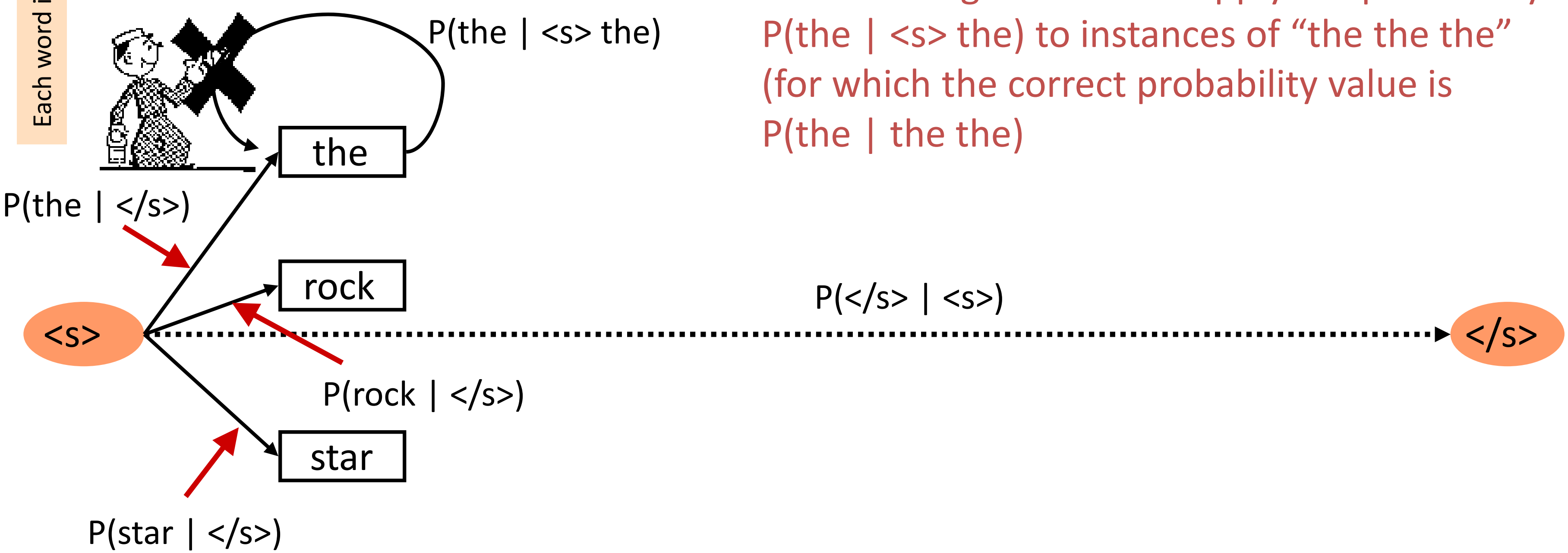
P(sing|sing sing)
sing
P(sing|<s> sing)
sing
P(song|sing sing)
P(</s>|sing sing)
P(song|<s> sing)
song
P(sing | <s>)
<s>
</s>
P(song|song sing)
P(sing|sing song)
P(sing|song sing)
P(song | <s>)
P(sing|<s> song)
sing
song
P(sing|song song)
P(song|sing song)
P(</s>|sing song)
P(song|<s> sing)
song
P(song|song song)

Trigram Representations



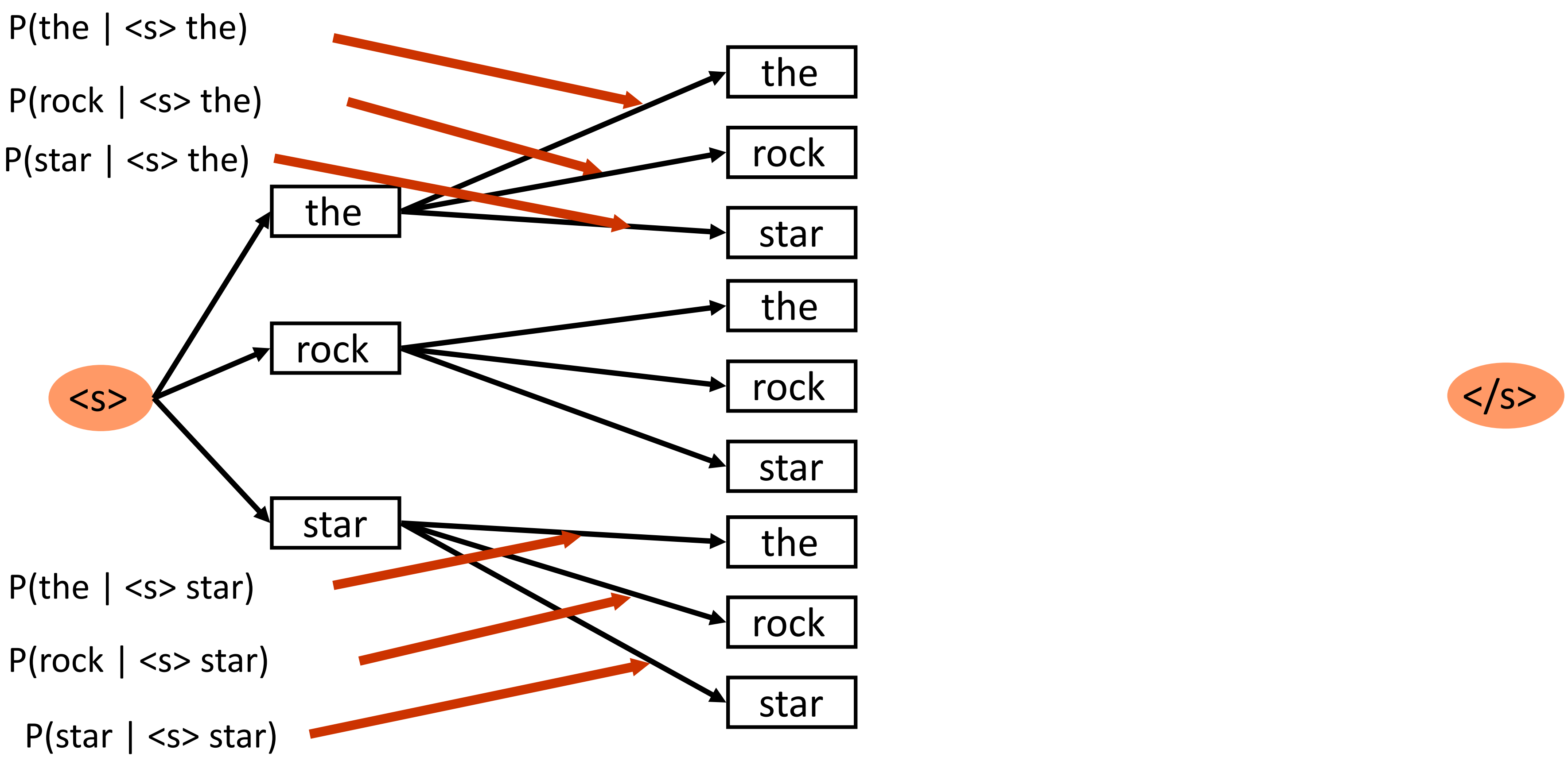
- Three word vocabulary “the”, “rock”, “star”
  - The graph initially begins with bigrams of <s>
  - There are edges from every node to “</s>”, that are not shown
  - Trigrams of “<s> the”..

This is wrong! This would apply the probability P(the | <s> the) to instances of “the the the” (for which the correct probability value is P(the | the the)



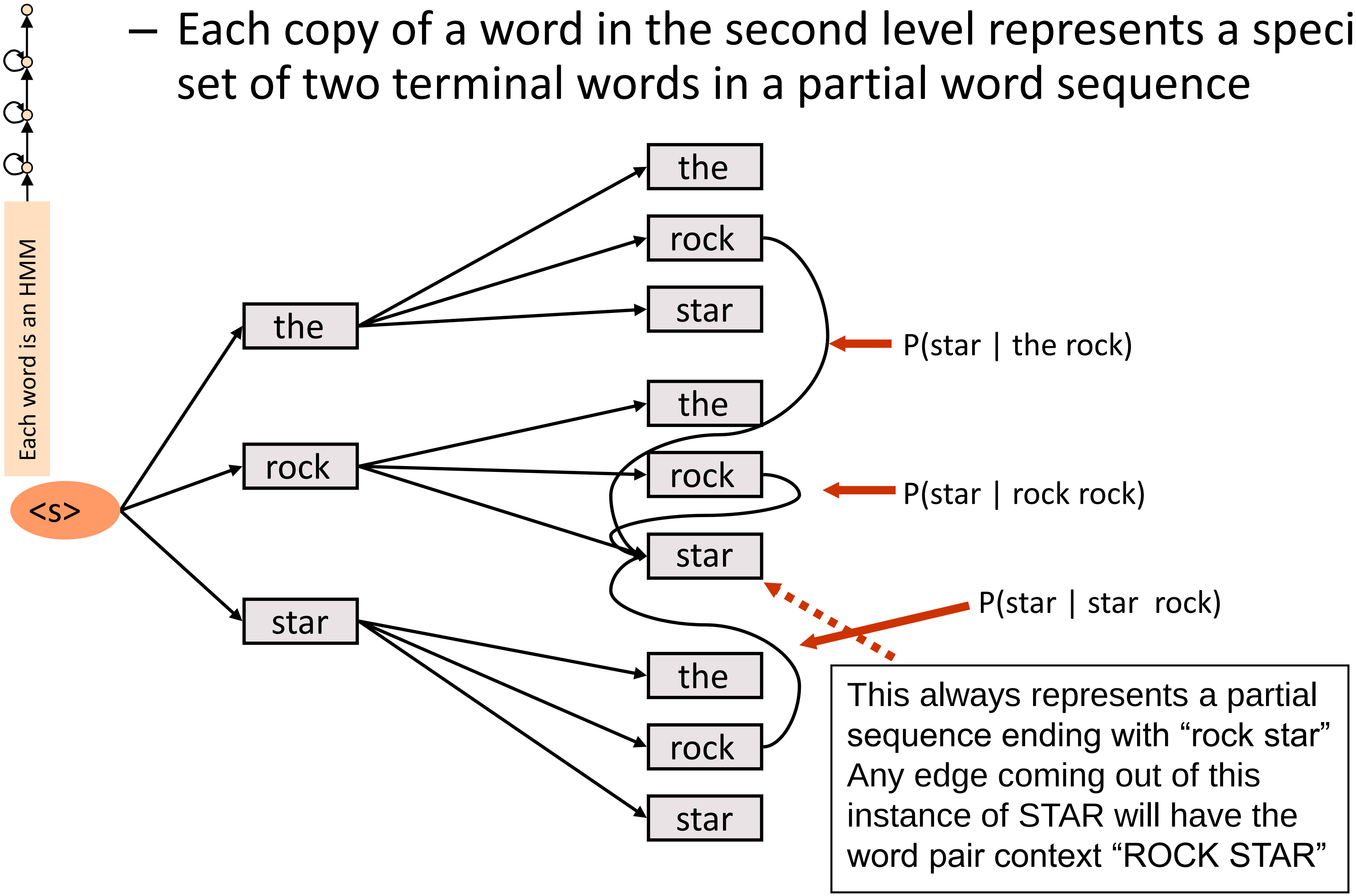
Trigram Representations

- Trigrams for all “<s> word” sequences
  - A new instance of every word is required to ensure that the two preceding symbols are “<s> word”

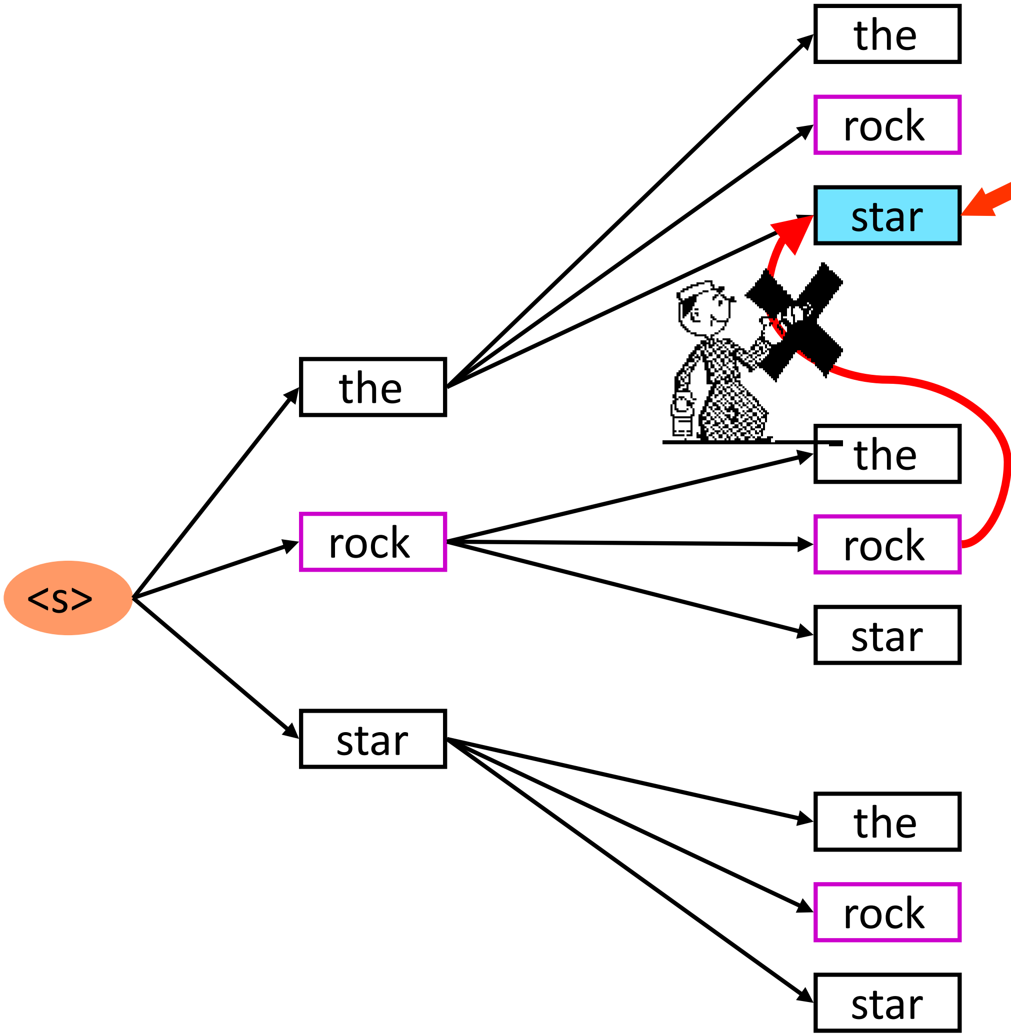


Trigram Representations

– Each copy of a word in the second level represents a specific set of two terminal words in a partial word sequence



Trigram Representations: Error



Edges coming out of this wrongly
connected STAR could have word
pair contexts that are either
“THE STAR” or “ROCK STAR”.
This is amibiguous. A word
cannot have incoming edges
from two or more different
words

Generic N-gram Representations

- The logic can be extended:
- A trigram decoding structure for a vocabulary of D words needs D word instances at the first level and D^2 word instances at the second level
  - Total of D(D+1) word models must be instantiated
  - Other, more expensive structures are also possible

- An N-gram decoding structure will need
  - $D + D^2 + D^3 \ldots D^{N-1}$ word instances
  - Arcs must be incorporated such that the exit from a word instance in the (N-1)th level always represents a word sequence with the same trailing sequence of N-1 words

ESTIMATING N-gram PROBABILITIES

Estimating N-gram Probabilities

- N-gram probabilities must be estimated from data
- Probabilities can be estimated simply by counting words in training text
- E.g. the training corpus has 1000 words in 50 sentences, of which 400 are "sing" and 600 are "song"
  - count(sing)=400; count(song)=600; count(</s>)=50
  - There are a total of 1050 tokens, including the 50 "end-of-sentence" markers
- UNIGRAM MODEL:
  - P(sing) = 400/1050; P(song) = 600/1050; P(</s>) = 50/1050
- BIGRAM MODEL: finer counting is needed. For example:
  - 30 sentences begin with sing, 20 with song
    - We have 50 counts of <s>
    - P(sing | <s>) = 30/50; P(song|<s>) = 20/50
  - 10 sentences end with sing, 40 with song
    - P(</s> | sing) = 10/400; P(</s>|song) = 40/600
  - 300 instances of sing are followed by sing, 90 are followed by song
    - P(sing | sing) = 300/400; P(song | sing) = 90/400;
  - 500 instances of song are followed by song, 60 by sing
    - P(song | song) = 500/600; P(sing|song) = 60/600

Estimating N-gram Probabilities

- Note that “</s>” is considered to be equivalent to a word. The probability for “</s>” are counted exactly like that of other words

- For N-gram probabilities, we count word sequences of length N
  - E.g. we count word sequences of length 2 for bigram LMs, and word sequences of length 3 for trigram LMs

- For N-gram probabilities of order N>1, we also count word sequences that include the word beginning and word end markers
  - E.g. counts of sequences of the kind “<s> w_a w_b” and “w_c w_d </s>”

- The N-gram probability of a word w_d given a context “w_a w_b w_c” is computed as
  - $P(w_d \mid w_a\, w_b\, w_c) = \text{Count}(w_a\, w_b\, w_c\, w_d) / \text{Count}(w_a\, w_b\, w_c)$
  - For unigram probabilities the denominator is simply the count of all word tokens (except the beginning of sentence marker <s>).
  - We do not explicitly compute the probability of P(<s>).

Estimating N-gram Probabilities

- Such direct estimation is however not possible in all cases
- E.g: 1000 word vocablary ➔ 1001*1001 possible bigrams
  - including the <s> and </s> markers
- Unlikely to encounter all 1002001 word pairs in any given training corpus
  - i.e. many of the corresponding bigrams will have 0 count
- However, these unseen bigrams may occur in *test* data
  - E.g., we may never see "sing sing" in the training corpus
  - P(sing | sing) will be estimated as 0
  - If a speaker says "sing sing" as part of any word sequence, at least the "sing sing" portion of it will never be recognized
- The problem gets worse as N increases
  - For a 1000 word vocabulary there are ~10^9 possible trigrams

Discounting

- We must assign a small non-zero probability to all N-grams that were never seen in the training data
- However, this means we will have to reduce the probability of other terms, to compensate
  - Example: We see 100 instances of sing, 90 of which are followed by sing, and 10 by </s>
  - The bigram probabilities computed directly are P(sing|sing) = 90/100, P(<s/>|sing) = 10/100
  - We never observed sing followed by song.
  - Let us attribute a small probability $\varepsilon > 0$ to P(song|sing)
  - But $90/100 + 10/100 + \varepsilon > 1.0$
  - To compensate we subtract a value α from P(sing|sing) and some value β from P(</s>|sing) such that
    - P(sing | sing) = 90 / 100 – α
    - P(</s> | sing) = 10 / 100 – β
    - P(sing | sing) + P(</s> | sing) + P(song | sing) = 90/100-α + 10/100-β + ε =1

Discounting and Smoothing

- The reduction of the probability estimates for seen N-grams, in order to assign non-zero probabilities to unseen N-grams is called discounting
  - Modifying probability estimates to be more generalizable is called *smoothing*
- Discounting and smoothing techniques:
  - Absolute discounting
  - Jelinek-Mercer smoothing
  - Good Turing discounting
  - Other methods
- All discounting techniques follow the same basic principle: they modify the *counts* of N-grams that are seen in the training data
  - The modification usually reduces the counts of seen N-grams
  - The withdrawn counts are reallocated to unseen N-grams
- Probabilities of seen N-grams are computed from the modified counts
  - The resulting N-gram probabilities are *discounted* probability estimates
  - Non-zero probability estimates are derived for unseen N-grams, from the counts that are reallocated to unseen N-grams

Absolute Discounting

- Subtract a constant from all counts
- E.g., we have a vocabulary of K words, $w_1, w_2, w_3 \ldots w_K$
- Unigram:
  - Count of word $w_i = C(i)$
  - Count of end-of-sentence markers (</s>) = C_{end}
  - Total count $C_{total} = \sum_i C(i) + C_{end}$
- Discounted Unigram Counts
  - $Cdiscount(i) = C(i) - \varepsilon$
  - $Cdiscount_{end} = C_{end} - \varepsilon$
- Discounted probability for seen words
  - $P(i) = Cdiscount(i) / C_{total}$
  - Note that the denominator is the total of the ***undiscounted*** counts
- If K_o words are seen in the training corpus, $K - K_o$ words are unseen
  - A total count of $K_o\varepsilon$, representing a probability $K_o\varepsilon / C_{total}$ remains unaccounted for
  - This is distributed among the $K - K_o$ words that were never seen in training
    - We will discuss how this distribution is performed later

Absolute Discounting: Higher order N-grams

- Bigrams: We now have counts of the kind
  - Contexts: Count(w1), Count(w2), … , Count(<s>)
    - Note <s> is also counted; but it is used ***only*** as a context
    - Context does ***not*** incoroporate </s>
  - Word pairs: Count (<s> w_1), Count(<s>,w_2),…,Count(<s> </s>),…, Count(w_1 w_1), …,Count(w_1 </s>) … Count(w_K w_K), Count(w_K </s>)
    - Word pairs ending in </s> are also counted

- Discounted counts:
  - DiscountedCount(w_i w_j) = Count(w_i w_j) – ε

- Discounted probability:
  - P(w_j | w_i) = DiscountedCount(w_i w_j) / Count(w_i)
  - Note that the discounted count is used only in the numerator

- For each context w_i, the probability $K_o(w_i)$ε / Count(w_i) is left over
  - $K_o(w_i)$ is the number of words that were seen following w_i in the training corpus
  - $K_o(w_i)$ε / Count(w_i) will be distributed over bigrams P(w_j | w_i), for words w_j such that the word pair w_i w_j was never seen in the training data

Absolute Discounting

- Trigrams: Word triplets and word pair contexts are counted
  - Context Counts: Count(<s> w_1), Count(<s> w_2), …
  - Word triplets: Count (<s> $w_1 w_1$),…, Count(w_K w_K, </s>)

- DiscountedCount(w_i w_j w_k) = Count(w_i w_j w_k) – ε

- Trigram probabilities are computed as the ratio of discounted word triplet counts and undiscounted context counts
- The same procedure can be extended to estimate higher-order N-grams

- **The value of ε:** The most common value for ε is 1
  - However, when the training text is small, this can lead to allocation of a disproportionately large fraction of the probability to unseen events
  - In these cases, ε is set to be smaller than 1.0, e.g. 0.5 or 0.1

- The optimal value of ε can also be derived from data
  - Via K-fold cross validation

K-fold cross validation to estimate ε

- Split training data into K equal parts
- Create K different groupings of the K parts by holding out one of the K parts and merging the rest of the K-1 parts together.
  - The held out part is a validation set, and the merged parts form a training set
  - This gives us K different partitions of the training data into training and validation sets
- For several values of ε
  - Compute K different language models with each of the K training sets
  - Compute the total probability Pvalidation(i) of the i^{th} validation set on the LM trained from the i^{th} training set
  - Compute the total probability $Pvalidation_e = Pvalidation(1)*Pvalidation(2)*..*Pvalidation(K)$
- Select the ε for which Pvalidation ε is maximum
- Retrain the LM using the *entire* training data, using the chosen value of ε

Jelinek Mercer smoothing

- Returns probability of an N-gram as a weighted combination of maximum likelihood N-gram and smoothed N-1 gram probabilities

$$P_{smooth}(word \mid w_a\ w_b\ w_c\ldots) = \lambda(w_a\ w_b\ w_c\ldots)P_{ML}(word \mid w_a\ w_b\ w_c\ldots) + \left(1.0 - \lambda(w_a\ w_b\ w_c\ldots)\right)P_{smooth}(word \mid w_b\ w_c\ldots)$$

- $P_{smooth}(word \mid w_a\ w_b\ w_c\ldots)$ is the N-gram probability used during recognition
  - The higher order (N-gram) term on the right hand side, $P_{ML}(word \mid w_a\ w_b\ w_c\ldots)$ is a maximum likelihood (counting-based) estimate
  - The lower order ((N-1)-gram term) $P_{smooth}(word \mid w_b\ w_c\ldots)$ is recursively obtained by interpolation between the ML estimate $P_{ML}(word \mid w_b\ w_c\ldots)$ and the smoothed estimate for the (N-2)-gram $P_{smooth}(word \mid w_c\ldots)$
  - All λ values lie between 0 and 1
  - Unigram probabilities are interpolated with a uniform probability distribution

Jelinek Mercer smoothing

$$P_{smooth}(word \mid w_a\ w_b\ w_c\ldots) = \lambda(w_a\ w_b\ w_c\ldots)P_{ML}(word \mid w_a\ w_b\ w_c\ldots) + \left(1.0 - \lambda(w_a\ w_b\ w_c\ldots)\right)P_{smooth}(word \mid w_b\ w_c\ldots)$$

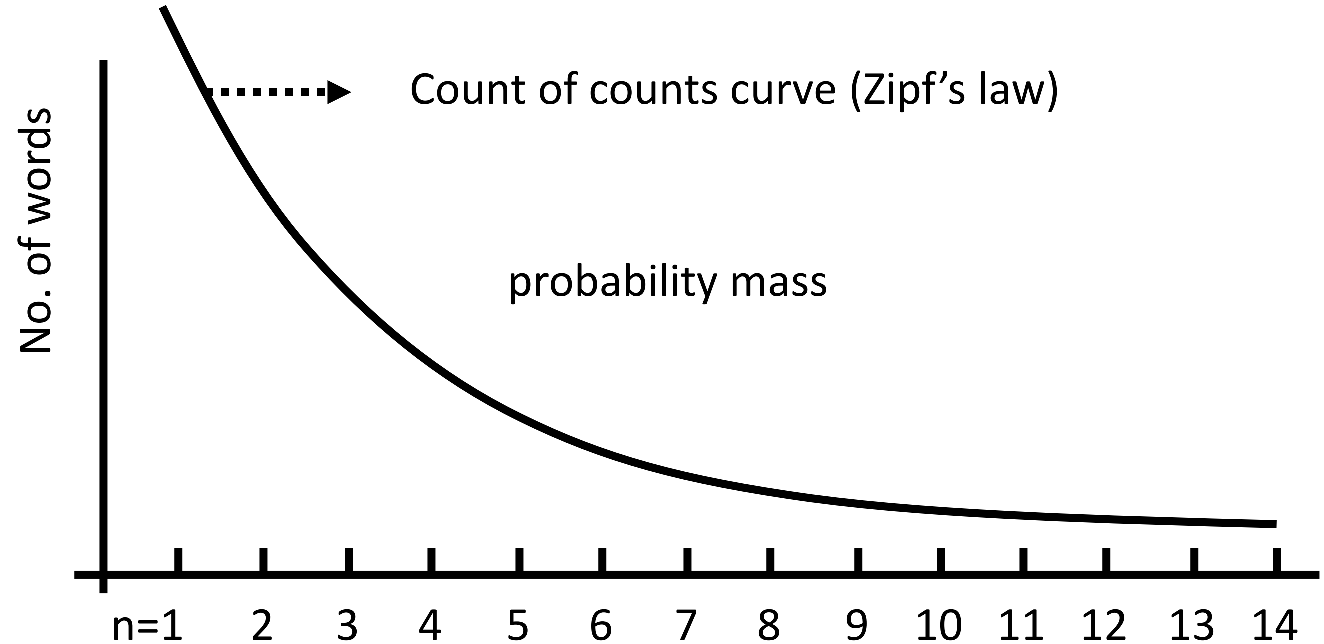
- The λ values must be estimated using held-out data
  - A combination of K-fold cross validation and the expectation maximization algorithms must be used
  - We will not present the details of the learning algorithm in this talk
  - Often, an arbitrarily chosen value of λ, such as λ = 0.5 is also very effective

Good Turing discounting: Zipf's law

- Zipf's law: The number of events that occur often is small, but the number of events that occur very rarely is very large.
- If n represents the number of times an event occurs in a unit interval, the number of events that occur n times per unit time is proportional to $1/n^{\alpha}$, where α is greater than 1
  - George Kingsley Zipf originally postulated that $\alpha = 1$.
  - Later studies have shown that a is $1 + \varepsilon$, where ε is slightly greater than 0
- Zipf's law is true for words in a language: the probability of occurrence of words starts high and tapers off. A few words occur very often while many others occur rarely.

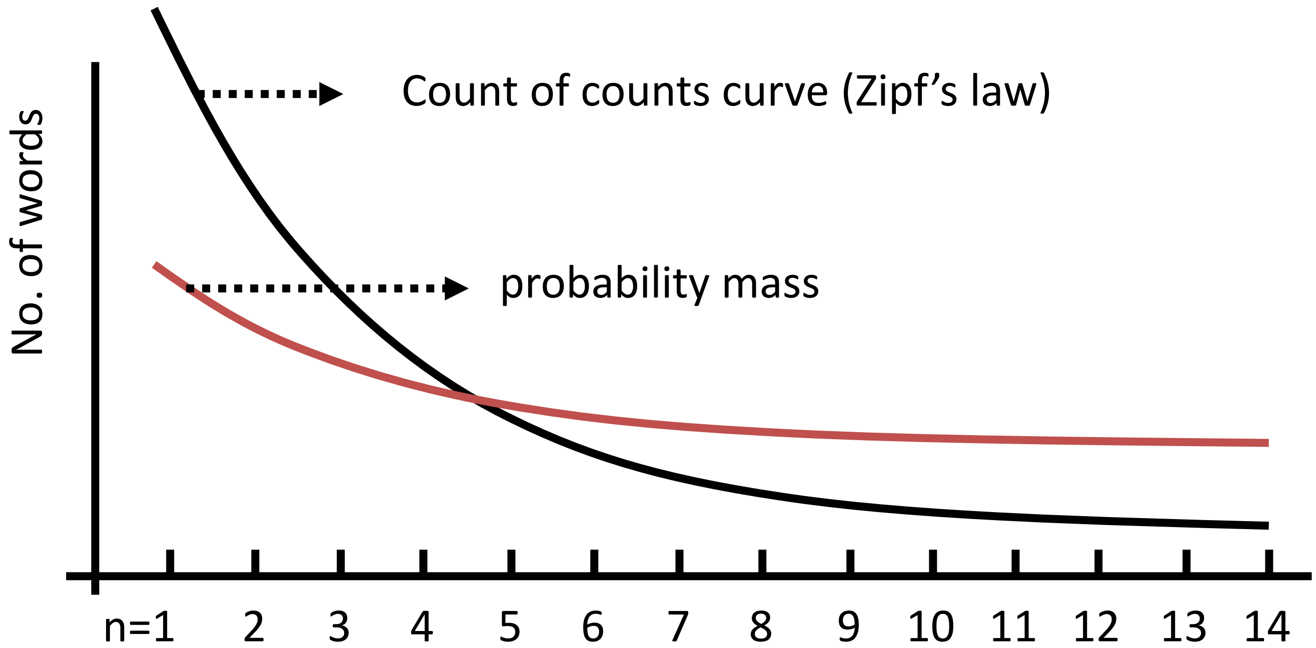
Zipf's law

- A plot of the count of counts of words in a training corpus typically looks like this:



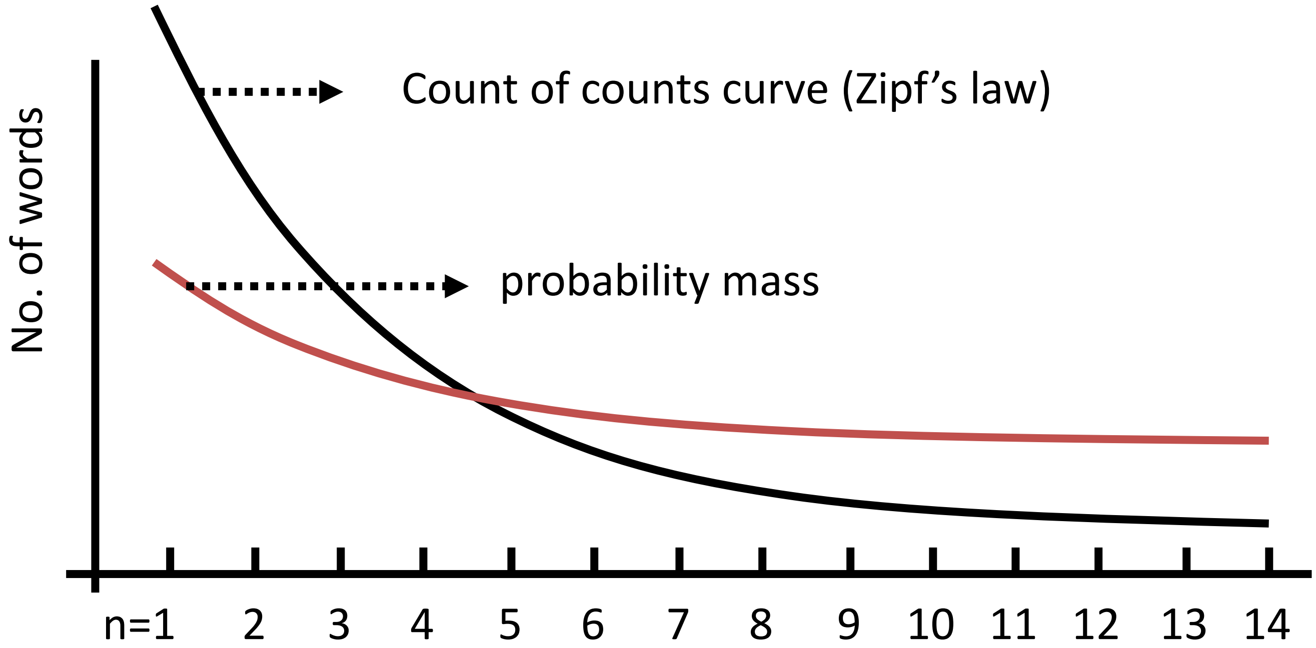
- In keeping with Zipf's law, the number of words that occur n times in the training corpus is typically more than the number of words that occur n+1 times

Total probability mass



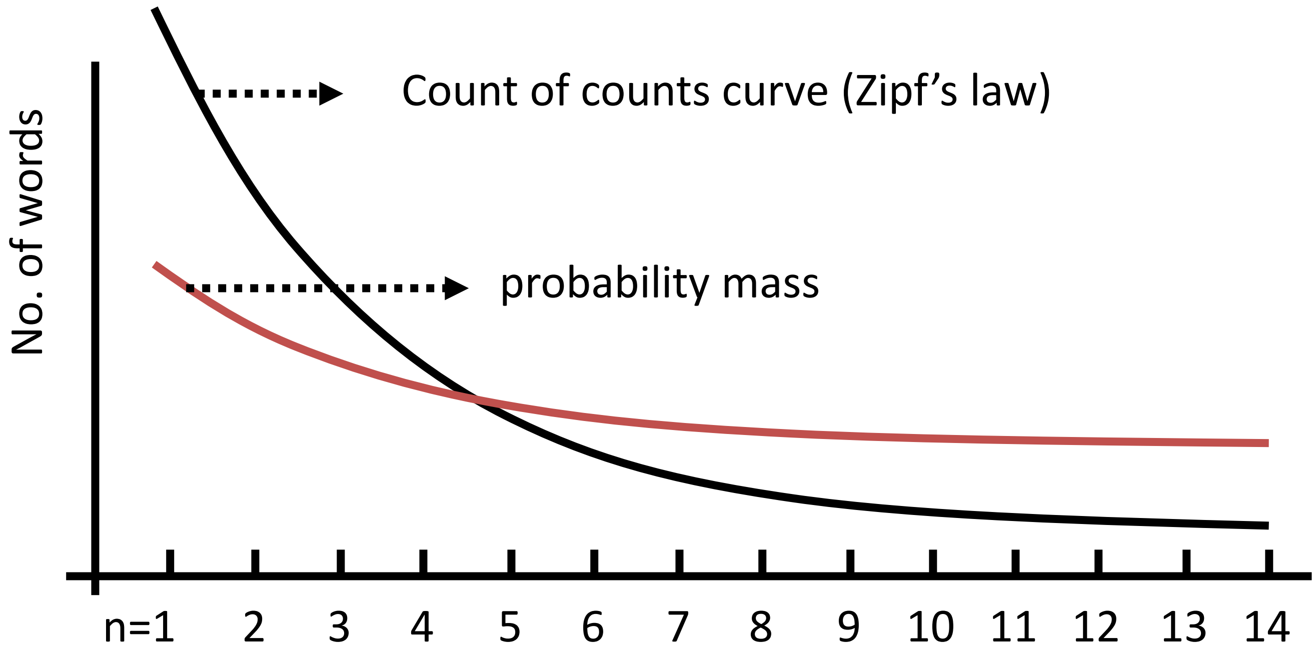
- Black line: Count of counts
  - Black line value at N = No. of words that occur N times
- Red line: Total probability mass of all events with that count
  - Red line value at 1 = (No. of words that occur once) / Total words
  - Red line value at 2 = 2 * (No. of words that occur twice) / Total words
  - Red line value at N = N * (No. of words that occur N times) / Total words

Total probability mass



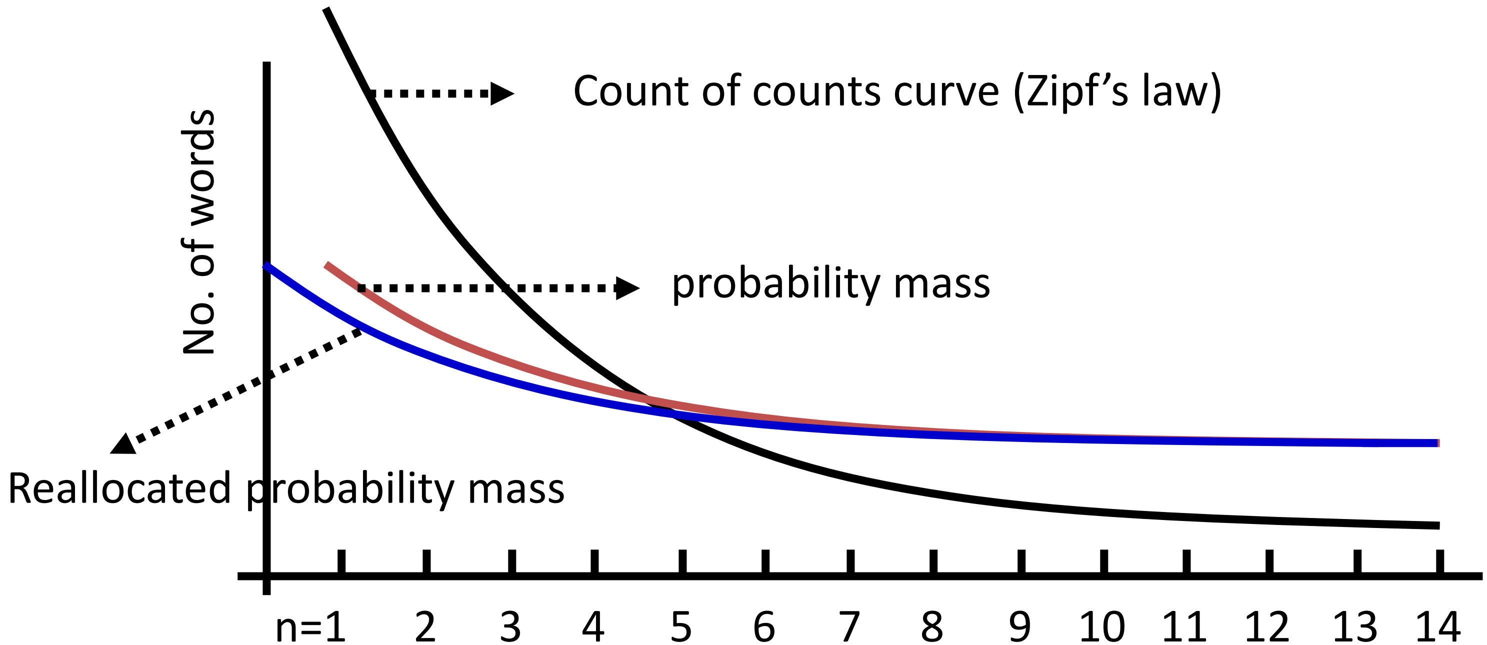
- Red Line
- $P(K) = K * N_K / N$
  - K = No. of times word was seen
  - N_K is no. of words seen K times
  - N: Total words

Good Turing Discounting



- In keeping with Zipf's law, the number of words that occur n times in the training corpus is typically more than the number of words that occur n+1 times
  - The total probability mass of words that occur n times falls slowly
  - Surprisingly, the total probability mass of rare words is greater than the total probability mass of common words, because of the large number of rare words

Good Turing Discounting

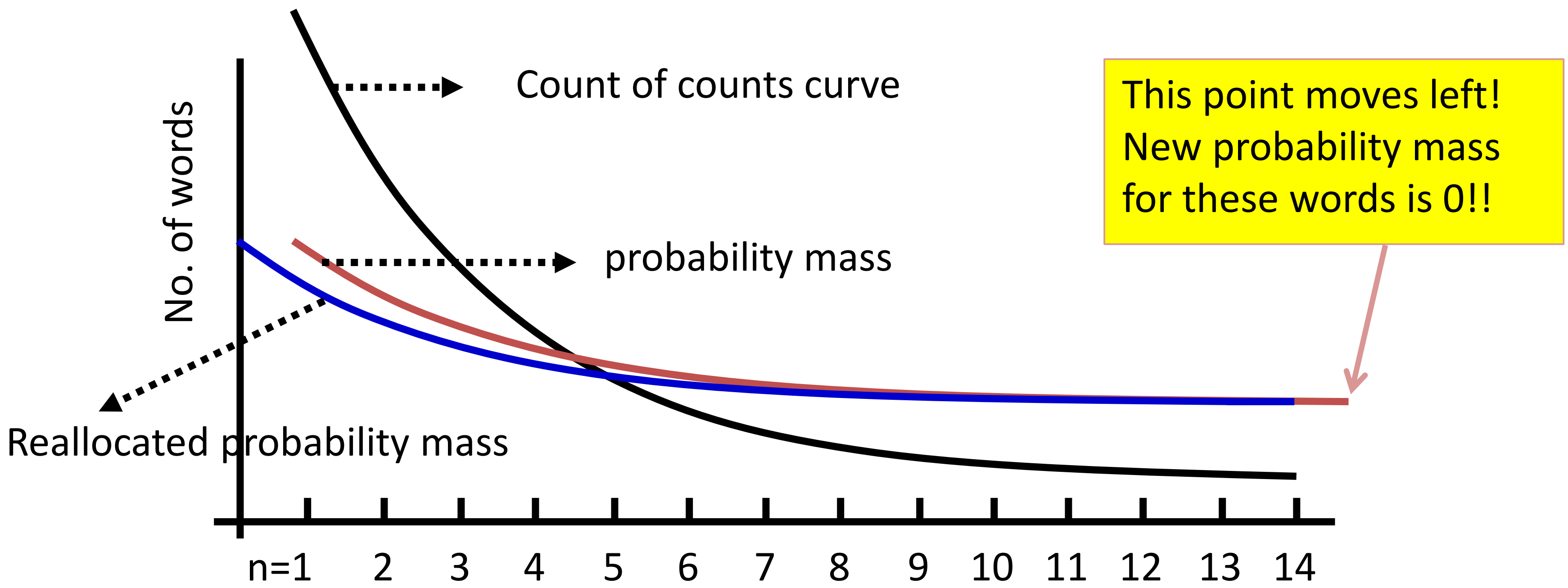


- Good Turing discounting reallocates probabilities
  - The total probability mass of all words that occurred n times is assigned to words that occurred n-1 times
  - The total probability mass of words that occurred once is reallocated to words that were never observed in training

Good Turing Discounting

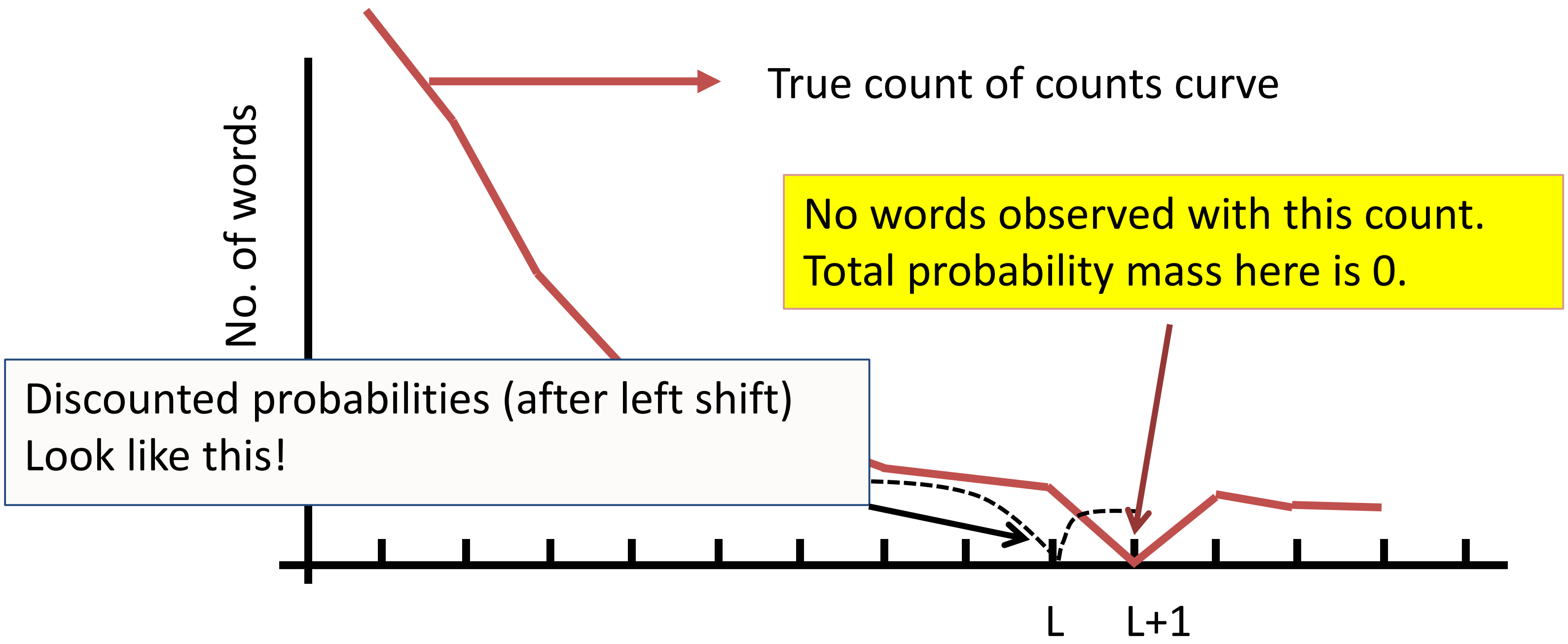
- Assign probability mass of events seen 2 times to events seen once.
  - Before discounting: P(word seen once) = $1 / N$
    - N = total words
  - After discounting:
    P(word seen once) = $(2*N_2 / N) / N_1$
    - N_2 is no. of words seen twice
    - N_1 is no. of words seen once
  - P(word seen once) = $(2*N_2 / N_1) / N$
- Discounted count for words seen once is:
  - $\mathbf{N_{1,discounted} = (2*N_2 / N_1)}$
  - Modified probability: Use discounted count as the count for the word

Good Turing Discounting



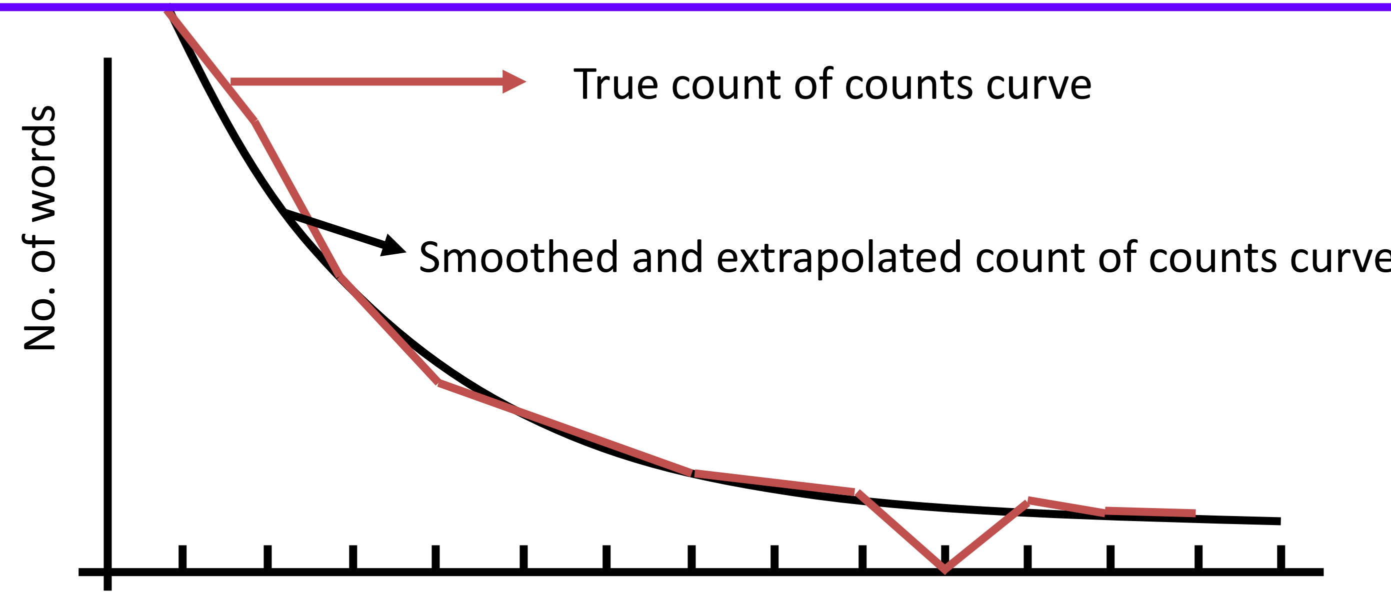
- The probability mass curve cannot simply be shifted left directly due to two potential problems
- Directly shifting the probability mass curve assigns 0 probability to the most frequently occurring words

Good Turing Discounting



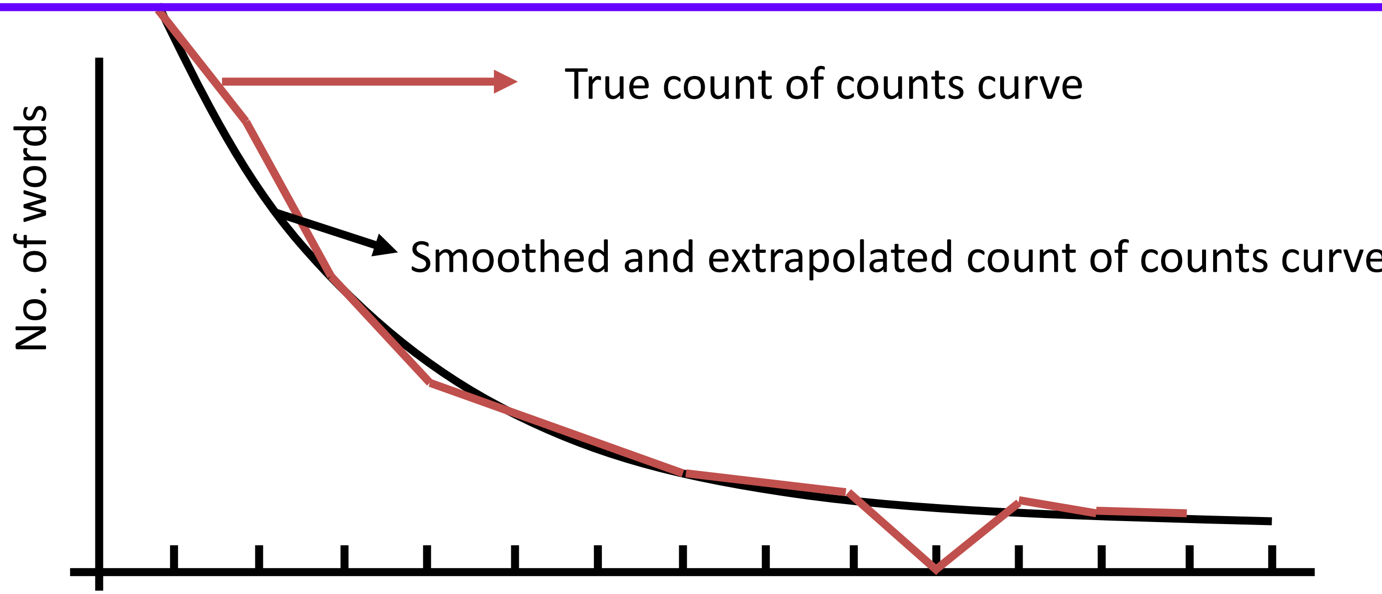
- The count of counts curve is often not continuous
  - We may have words that occurred L times, and words that occurred L+2 times, but none that ocurred L+1 times
  - By simply reassigning probability masses backward, words that occurred L times are assigned the total probability of words that ocurred L+1 times = 0!

Good Turing Discounting



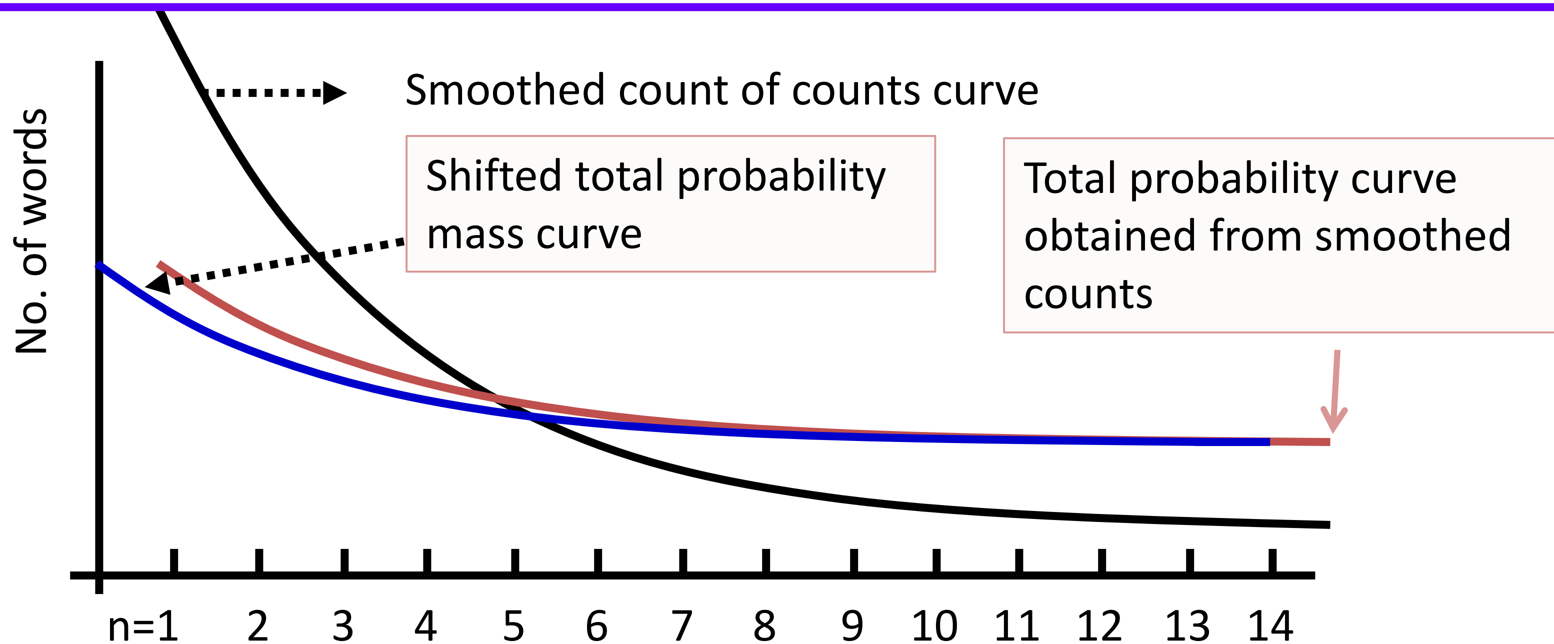
- The count of counts curve is smoothed and extrapolated
  - Smoothing fills in "holes" – intermediate counts for which the curve went to 0
  - Smoothing may also vary the counts of events that were observed
  - Extrapolation extends the curve to one step beyond the maximum count observed in the data
- Smoothing and extrapolation can be done by linear interpolation and extrapolation, or by fitting polynomials or splines
- Probability masses are computed from the smoothed count-of-counts and reassigned

Good Turing Discounting



- **Step 1:** Compute count-of-counts curve
  - Let r(i) be the number of words that occurred i times
- **Step 2:** Smooth and extend count-of-count curve
  - Let r'(i) be the smoothed count of the number of words that occurred i times.
- The total smoothed count of all words that occurred i times is r'(i) * i.
  - We operate entirely with the smoothed counts from here on

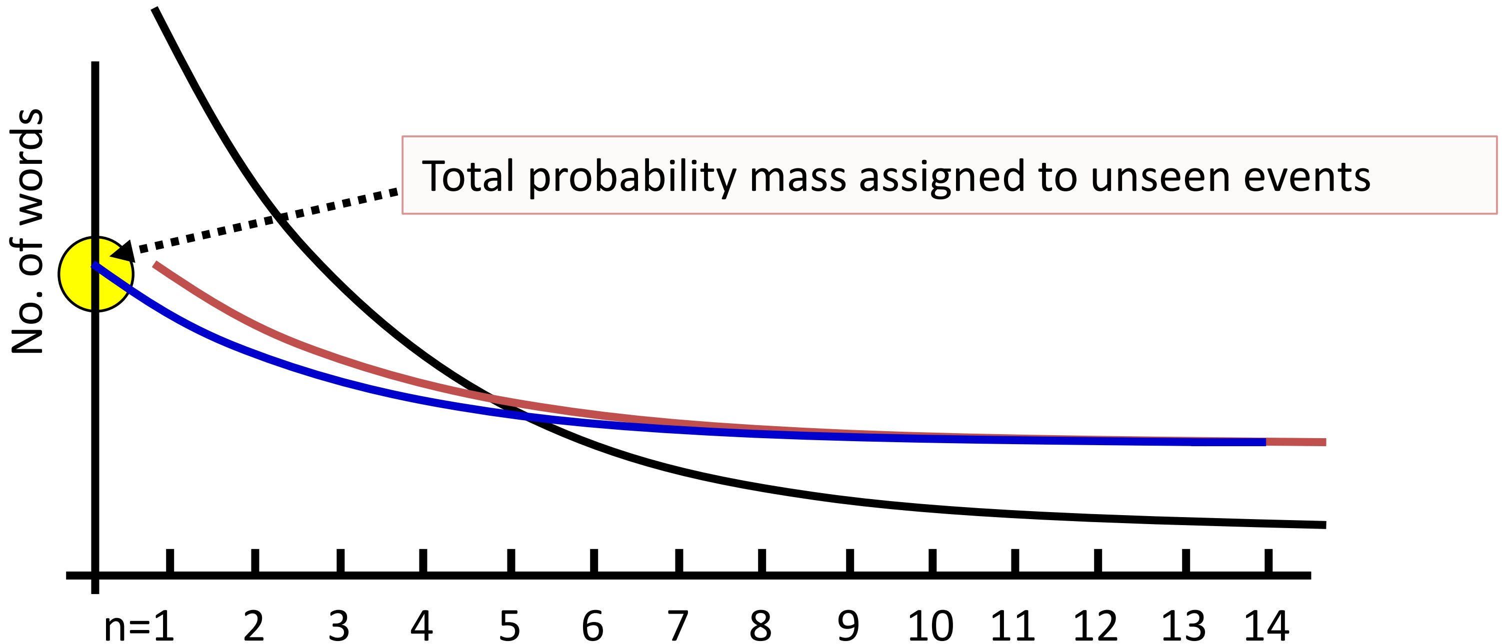
Good Turing Discounting



- **Step 3:** Reassign total smoothed counts r'(i)*i to words that occurred i-1 times.
  - reassignedcount(i-1) = r'(i)*i / r'(i-1)
- **Step 4:** Compute modified total count from smoothed counts
  - totalreassignedcount = Σ_i smoothedprobabilitymass(i)
- **Step 5:** A word *w* with count *i* is assigned probability

  P(*w*/ *context*) = reassignedcount(i) / totalreassignedcount

Good Turing Discounting



- **Step 6:** Compute a probability for unseen terms!!!!

- A probability mass $P_{leftover}$ = r'(1)*N_1 / totalreassignedcount is left over
  - Reminder: r'(1) is the smoothed count of words that occur once
  - The left-over probability mass is reassigned to words that were not seen in the training corpus

- **P(any unseen word) = $P_{leftover}$ / N_{unseen}**

Good Turing estimation of LM probabilities

- UNIGRAMS:
  - The count-of-counts curve is derived by counting the words (including </s>) in the training corpus
  - The count-of-counts curve is smoothed and extrapolated
  - Word probabilities are computed for observed words are computed from the smoothed, reassigned counts
  - The left-over probability is reassigned to unseen words

- BIGRAMS:
  - For each word context W, (where W can also be <s>), the same procedure given above is followed: the count-of-counts for all words that occur immediately after W is obtained, smoothed and extrapolated, and bigram probabilities for words seen after W are computed.
  - The left-over probability is reassigned to the bigram probabilities of words that were never seen following W in the training corpus

- Higher order N-grams: The same procedure is followed for every word context W_1 W_2… W_{N-1}

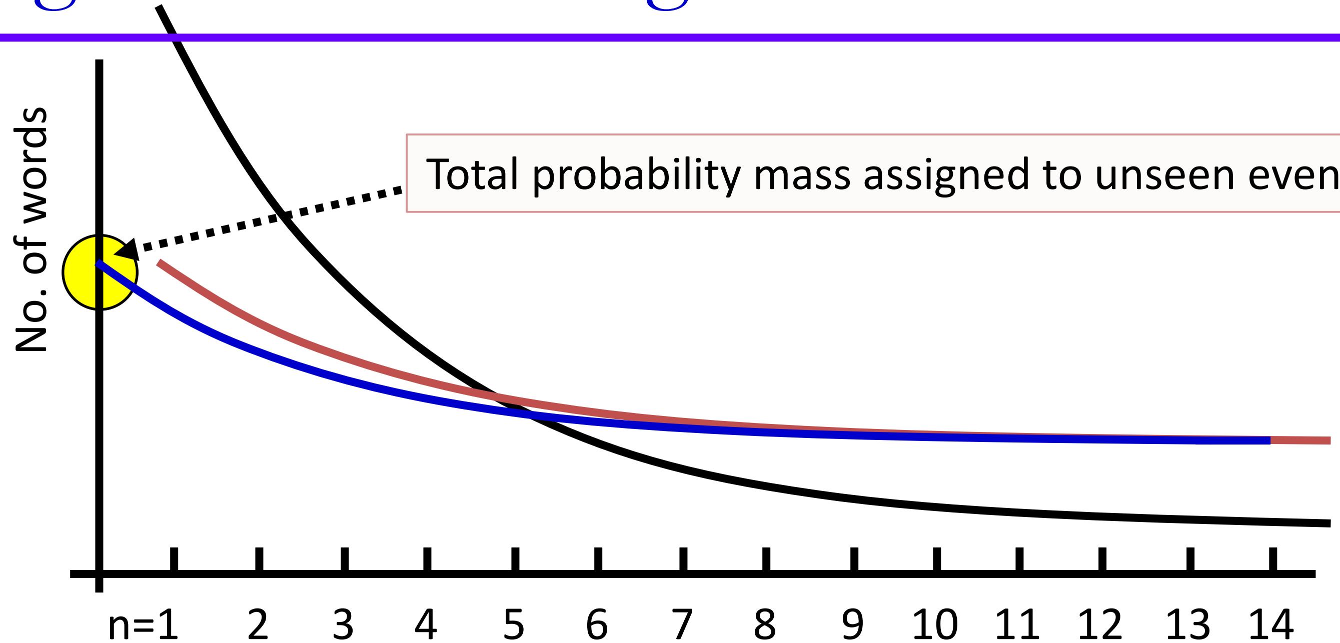
Reassigning left-over probability to unseen words

- All discounting techniques result in a some left-over probability to reassign to unseen words and N-grams

- For unigrams, this probability is uniformly distributed over all unseen words
  - The vocabulary for the LM must be prespecified
  - The probability will be reassigned uniformly to words from this vocabulary that were not seen in the training corpus

- For higher-order N-grams, the reassignment is done differently
  - Based on lower-order N-gram, i.e. (N-1)-gram probabilities
  - The process by which probabilities for unseen N-grams is computed from (N-1)-gram probabilities is referred to as "backoff"

Dealing with unseen Ngrams



- UNIGRAMS: A probability mass $P_{leftover} = r'(1)*N_1$ / totalreassignedcount is left over and distributed ***uniformly*** over unseen words
  - **P(any unseen word) = $P_{leftover} / N_{unseen}$**

- BIGRAMS: We only count over all words in a particular context
  - E.g. all words that followed word "w3"
  - We count words and smooth word counts only over this set (e.g. words that followed w3)
  - We can use the same discounting principle as above to compute probabilities of unseen bigrams of w3 (i.e bigram probabilities that a word will follow w3, although it was never observed to follow w3 in the training set)
  - **CAN WE DO BETTER THAN THIS?**

Unseen N-grams : Backoff

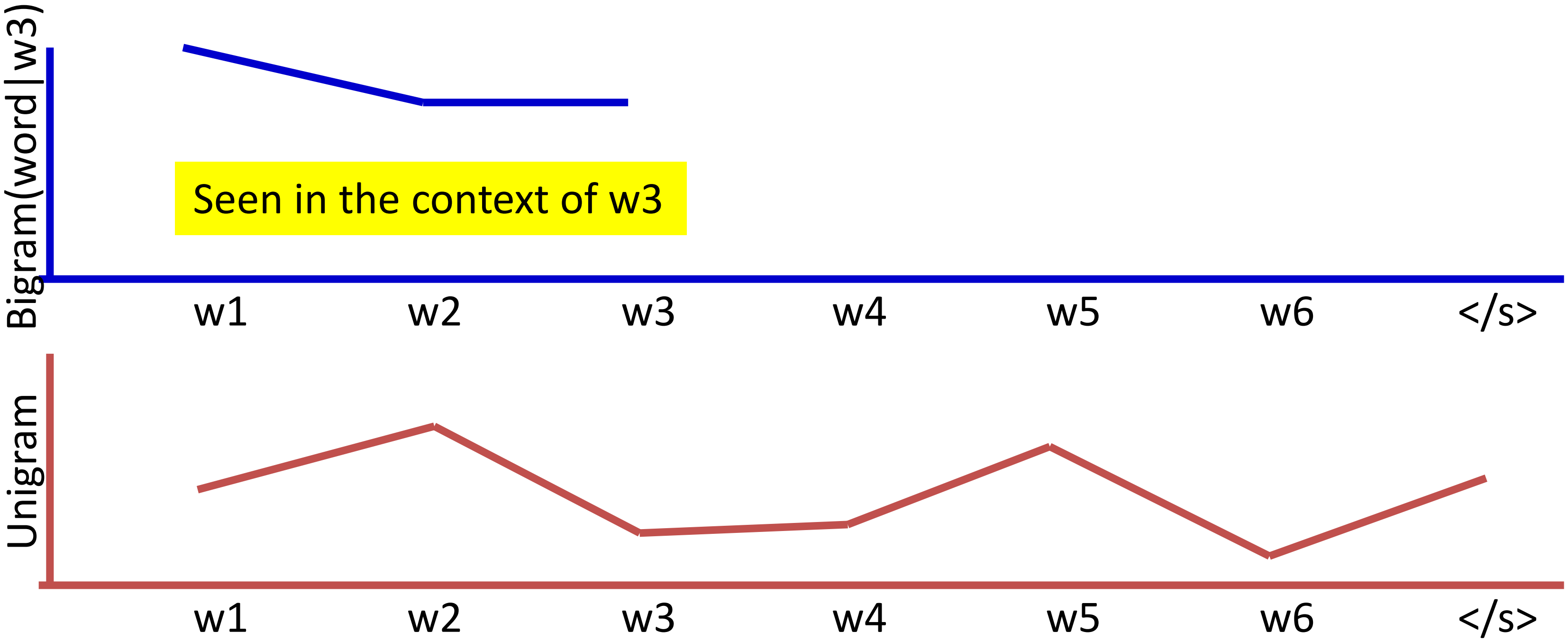
- Example: Words w5 and w6 were never observed to follow w3 in the training data
  - E.g. we never saw “dog” or “bear” follow the word “the”

- Backoff assumption: Relative frequencies of w5 and w6 will be the same in the context of w3 (bigram) as they are in the language in general (Unigrams)
  - If the number of times we saw “dog” in the entire training corpus was 10x the no. of times we saw “bear”, then we assume that the number of times we will see “dog” after “the” is also 10x the no. of times we will see “bear” after “the”

- Generalizing: N-gram probabilities of words that are never seen (in the training data) in the given N-gram context follow the same distribution pattern observed in the N-1 gram context

N-gram LM : Backoff

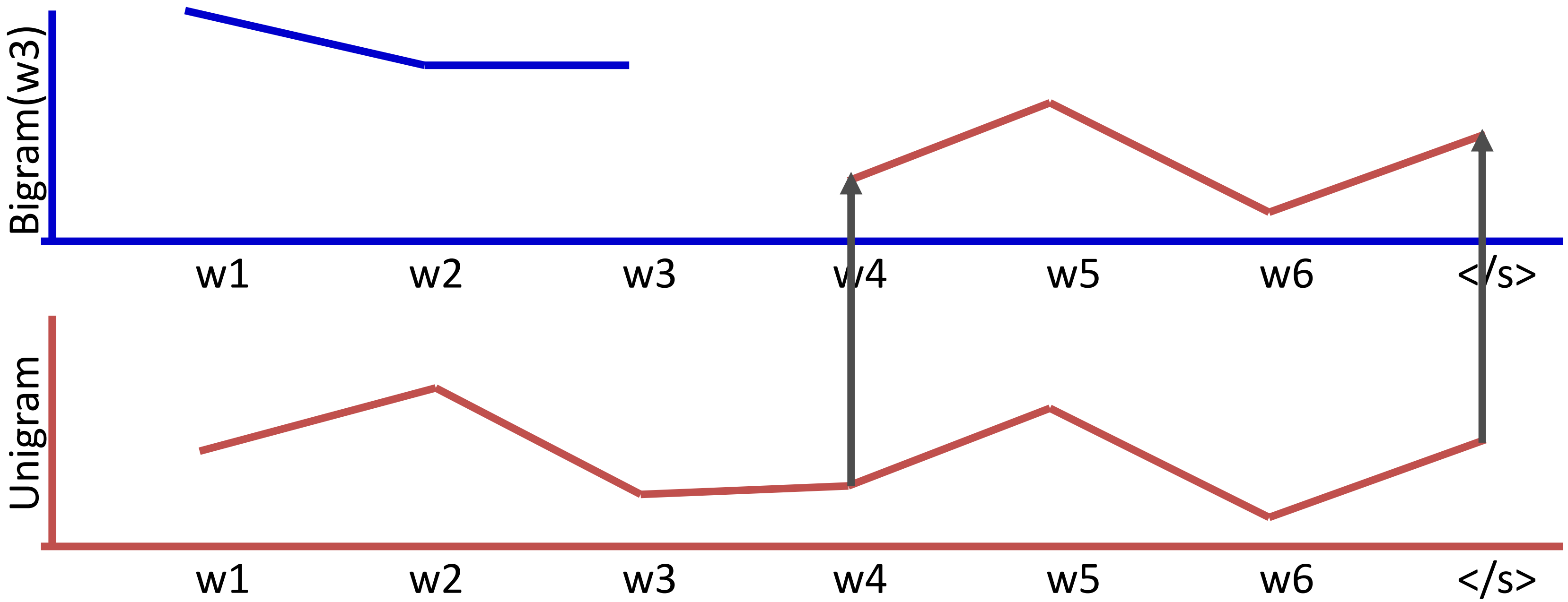
- Explanation with a bigram example



- Unigram probabilities are computed and known before bigram probabilities are computed
- Bigrams for P(w1 | w3), P(w2 | w3) and P(w3 | w3) were computed from discounted counts. w4, w5, w6 and </s> were never seen after w3 in the training corpus

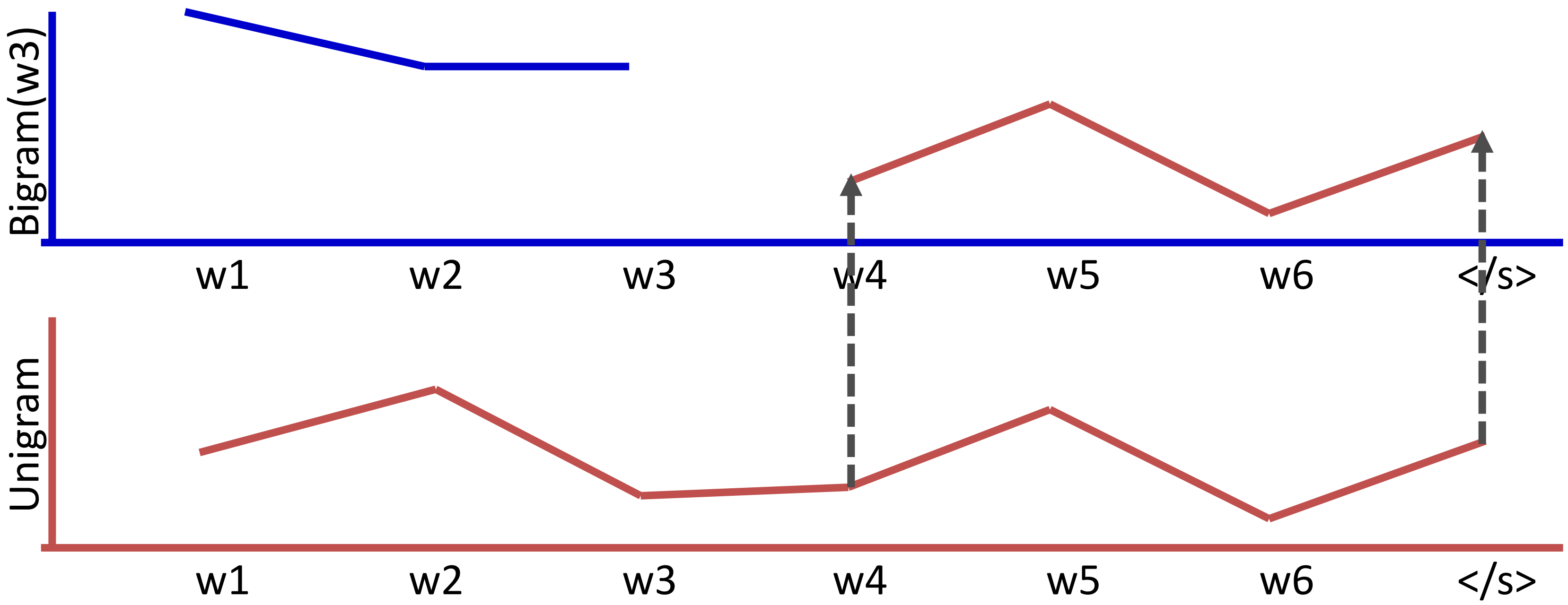
N-gram LM : Backoff

- Explanation with a bigram example



- The probabilities P(w4|w3), P(w5|w3), P(w6|w3) and P(</s>|w3) are assumed to follow the same pattern as the unigram probabilities P(w4), P(w5), P(w6) and P(</s>)
- They must, however be scaled such that
  P(w1|w3) + P(w2|w3) + P(w3|w3) + scale*(P(w4)+P(w5)+P(w6)+P(</s>)) = 1.0
- The *backoff* bigram probability for the unseen bigram P(w4 | w3) = scale*P(w4)

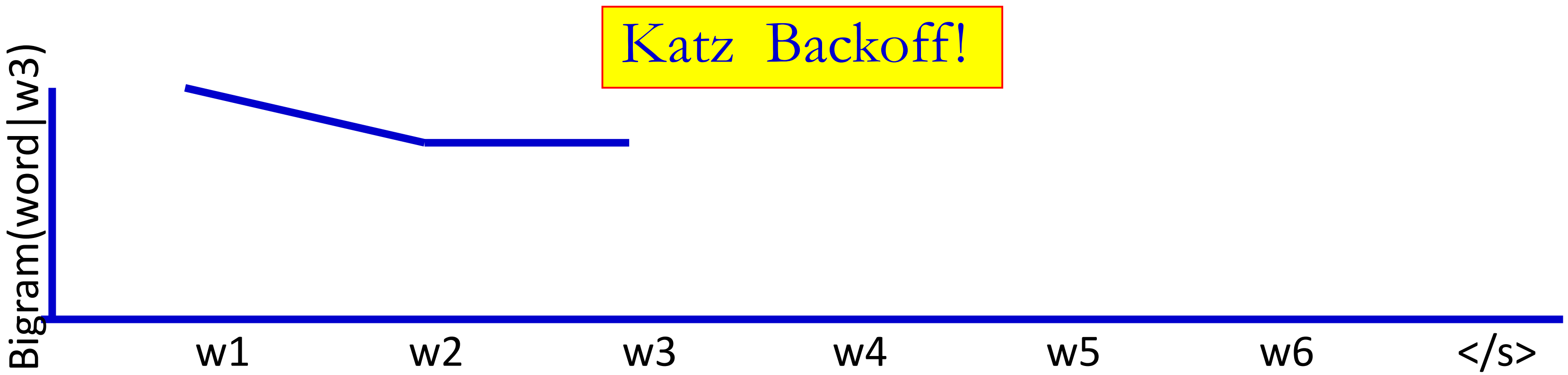
N-gram LM : Backoff



- P(w1|w3) + P(w2|w3) + P(w3|w3) + scale*(P(w4)+P(w5)+P(w6)+P(</s>)) = 1.0
- The *backoff* bigram probability for the unseen bigram P(w4 | w3) = scale*P(w4)
- The *scale* term is called the backoff term. It is specific to w3
  - Scale = backoff(w3)
  - Specificity is because the various terms used to compute scale are specific to w3

$$backoff(w_3) = \frac{1 - P(w_1 | w_3) - P(w_2 | w_3) - P(w_3 | w_3)}{P(w_4) + P(w_5) + P(w_6) + P(</s>)}$$

N-gram LM : Backoff from N-gram to N-1 gram



- Assumption: When estimating N-gram probabilities, we already have access to all N-1 gram probabilities
- Let $w_1 \ldots w_K$ be the words in the vocabulary (includes </s>)
- Let $\mathbf{W}_{N-1}$ be the context for which we are trying to estimate N-gram probabilities
  - Will be some sequence of N-1 words (for N-gram probabilities)
  - i.e we wish to compute all probabilities P(word | $\mathbf{W}_{N-1}$)
  - E.g $\mathbf{W_3}$ = "w_a w_b w_c". We wish to compute all 4-gram probabilities P(word | w_a w_b w_c)

N-gram LM : Backoff from N-gram to N-1 gram

- **Step 1:** Compute leftover probability mass for unseen N-grams (of the form P(word| $\mathbf{W}_{N-1}$)) using Good Turing discounting
  - $P_{leftover}(\mathbf{W}_{N-1})$ – this is specific to context $\mathbf{W}_{N-1}$ as we are only counting words that follow word sequence $\mathbf{W}_{N-1}$

- **Step 2:** Compute backoff weight

$$backoff(\mathbf{W}_{N-1}) = \frac{1 - \sum_{w \text{ was seen following } \mathbf{W}_{N-1} \text{ in the training text}} P(w|\mathbf{W}_{N-1})}{\sum_{w \text{ was NOT seen following } \mathbf{W}_{N-1} \text{ in the training text}} P(w|\mathbf{W}_{N-2})}$$

  - Note $\mathbf{W}_{N-2}$ in the denominator. If $\mathbf{W}_{N-1}$ is "$w_a\ w_b\ w_c$", $\mathbf{W}_{N-2}$ is "$w_b\ w_c$"
    - The trailing N-2 words only
    - We already have N-1 gram probabilities of the form P(w | $\mathbf{W}_{N-2}$)

- **Step 3:** We can now compute N-gram probabilities for unseen Ngrams

$$P(w|\mathbf{W}_{N-1}) = backoff(\mathbf{W}_{N-1})P(w|\mathbf{W}_{N-2})$$

  - Actually, this is done "on demand" – there's no need to store them explicitly.

Backoff is recursive

- In order to estimate the backoff weight needed to compute N-gram probabilities for unseen N-grams, the corresponding N-1 grams are required (as in the following 4-gram example)

$$P(w \mid w_a w_b w_c) = backoff(w_a w_b w_c) P(w \mid w_b w_c)$$

  - The corresponding N-1 grams might also not have been seen in the training data

- If the backoff N-1 grams are also unseen, they must in turn be computed by backing off to N-2 grams
  - The backoff weight for the unseen N-1 gram must also be known
  - i.e. it must also have been computed already

- before higher-order N-gram parameters can be estimatedAll lower order N-gram parameters (including probabilities and backoff weights) must be computed

Learning backoff N-gram models

- First compute Unigrams
  - Count words, perform discounting, estimate discounted probabilities for all seen words
  - Uniformly distribute the left-over probability over unseen unigrams
- Next, compute bigrams. For each word W seen in the training data:
  - Count words that follow that W. Estimate discounted probabilities P(word | W) for all words that were seen after W.
  - Compute the backoff weight b(W) for the context W.
  - The set of explicity estimated P(word | W) terms, and the backoff weight b(W) together permit us to compute all bigram probabilities of the kind: P(word | W)
- Next, compute trigrams: For each word pair "w_a w_b" seen in the training data:
  - Count words that follow that "w_a w_b". Estimate discounted probabilities P(word | wa wb) for all words that were seen after "wa wb".
  - Compute the backoff weight b(w_a w_b) for the context "w_a w_b".
- The process can be continued to compute higher order N-gram probabilities.

Contents of a completely trained N-gram backoff model

- Unigram probabilities for all words in the vocabulary
- Backoff weights for all words in the vocabulary
- Bigram probabilities for some, but not all bigrams
  - i.e. for all bigrams that were seen in the training data
- If N>2, then: backoff weights for all seen word pairs
  - If the word pair was never seen in the training corpus, it will not have a backoff weight. The backoff weight for all word pairs that were not seen in the training corpus is implicitly set to 1
- …
- N-gram probabilities for some, but not all N-grams
  - N-grams seen in training data
- Note that backoff weights are not required for N-length word sequences in an N-gram LM
  - Since backoff weights for N-length word sequences are only useful to compute backed off N+1 gram probabilities

Backoff trigram LM: An example

```
\1-grams:
-1.2041 <UNK>      0.0000          0.0625   <UNK>
-1.2041 </s>       0.0000          0.0625   </s>
-1.2041 <s>       -0.2730          0.0625   <s>          0.5333
-0.4260 one       -0.5283          0.3750   one
-1.2041 three     -0.2730          0.0625   three
-0.4260 two       -0.5283          0.3750   two
\2-grams:
-0.1761 <s> one      0.0000
-0.4771 one three    0.1761
-0.3010 one two      0.3010
-0.1761 three two    0.0000          (1-10^(-0.1761))/(1-0.3750)=0.533
-0.3010 two one      0.3010
-0.4771 two three    0.1761
\3-grams:
-0.3010 <s> one two
-0.3010 one three two
-0.4771 one two one
-0.4771 one two three
-0.3010 three two one
-0.4771 two one three
-0.4771 two one two
-0.3010 two three two
```

Obtaining N-gram probability from backoff N-gram LM

- To retrieve a probability P(word | w_a w_b w_c …)
  - How would a function written for returning N-gram probabilities work?

- Look for the probability P(word | w_a w_b w_c …) in the LM
  - If it is explicitly stored, return it

- If P(word | w_a w_b w_c …) is not explicitly stored in the LM retrive it by backoff to lower order probabilities:
  - Retrieve backoff weight backoff(w_a w_b w_c..) for word sequence wa wb wc
    - If it is stored in the LM, return it
    - Otherwise return 1
  - Retrieve P(word | w_b w_c …) from the LM
    - If P(word | wb wc ..) is not explicitly stored in the LM, derive it backing off
    - This will be a recursive procedure
  - Return P(word | w_b w_c …) * backoff(w_a w_b w_c..)

Toolkits for training Ngram LMs

- CMU-Cambridge LM Toolkit
- SRI LM Toolkit
- MSR LM toolkit
  - Good for large vocabularies
- Many many others..
- ..

- Your own toolkit here

Training a language model using CMU-Cambridge LM toolkit

Contents of textfile

<s> the term cepstrum was introduced by Bogert et al and has come to be
accepted terminology for the
inverse Fourier transform of the logarithm of the power spectrum
of a signal in nineteen sixty three Bogert Healy and Tukey published a paper
with the unusual title
The Quefrency Analysis of Time Series for Echoes Cepstrum Pseudoautocovariance
Cross Cepstrum and Saphe Cracking
they observed that the logarithm of the power spectrum of a signal containing an
echo has an additive
periodic component due to the echo and thus the Fourier transform of the
logarithm of the power
spectrum should exhibit a peak at the echo delay
they called this function the cepstrum
interchanging letters in the word spectrum because
in general, we find ourselves operating on the frequency side in ways customary
on the time side and vice versa
Bogert et al went on to define an extensive vocabulary to describe this new
signal processing technique however only the term cepstrum has been widely used
The transformation of a signal into its cepstrum is a homomorphic transformation
and the concept of the cepstrum is a fundamental part of the theory of homomorphic
systems for processing signals that have been combined by convolution
</s>

vocabulary

<s>
</s>
the
term
cepstrum
was
introduced
by
Bogert
et
al
and
has
come
to
be
accepted
terminology
for
inverse
Fourier
transform
of
logarithm
Power
. . .

Contents of contextfile

<s>

Training a language model using CMU-Cambridge LM toolkit

To train a bigram LM (n=2):

$bin/text2idngram -vocab vocabulary -n 2 -write_ascii < textfile > idngm.tempfile

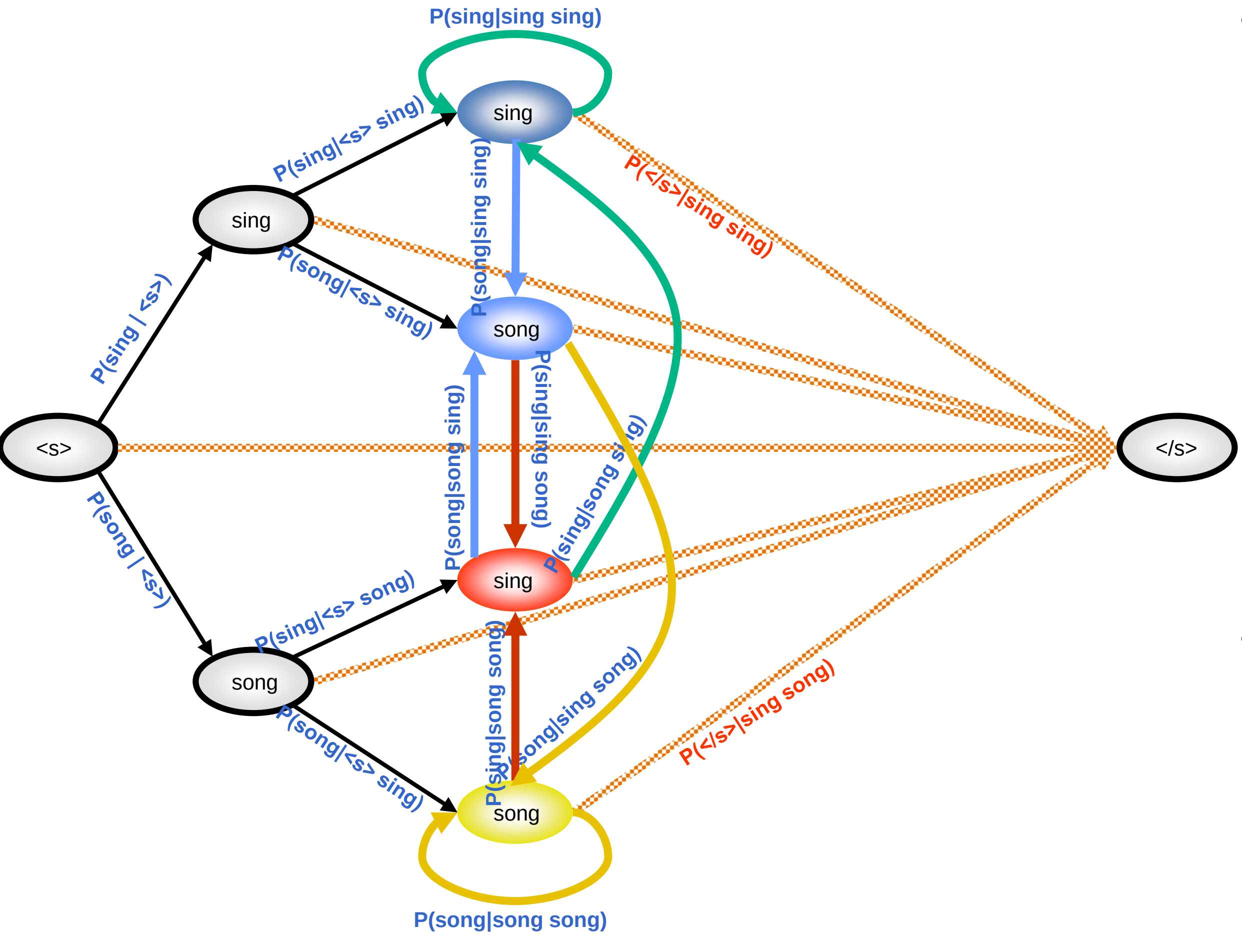
$bin/idngram2lm -idngram idngm.tempfile -vocab vocabulary -arpa MYarpaLM -context contextfile -absolute -ascii_input -n 2 (optional: -cutoffs 0 0 or –cutoffs 1 1 ….)

OR

$bin/idngram2lm -idngram idngm.tempfile -vocab vocabulary -arpa MYarpaLM -context contextfile -good_turing -ascii_input -n 2

….

Representing N-gram LMs as graphs



- For recognition, the N-gram LM can be represented as a finite state graph
  - Recognition can be performed exactly as we would perform recognition with grammars
- Problem: This graph can get enormously large
  - There is an arc for every single N-gram probability!
  - Also for every single N-1, N-2 .. 1-gram probabilities

The representation is wasteful

- In a typical N-gram LM, the vast majority of bigrams, trigrams (and higher-order N-grams) are computed by backoff
  - They are not seen in training data, however large

$$P(w \mid w_a w_b w_c) = backoff(w_a w_b w_c) P(w \mid w_b w_c)$$

- The backed-off probability for an N-gram is obtained from the N-1 gram!

- So for N-grams computed by backoff it should be sufficient to store only the N-1 gram in the graph
  - Only have arcs for $P(w \mid w_b\, w_c)$; not necessary to have explicit arcs for $P(w \mid w_a\, w_b\, w_c)$
  - This will reduce the size of the graph ***greatly***

N-gram LM as FSGs: Accounting for backoff

- N-Gram language models with back-off can be represented as finite state grammars
  - That explicitly account for backoff!

- This also permits us to use grammar-based recognizers to perform recognition with Ngram LMs

- There are a few precautions to take, however

N-gram to FSG conversion: Trigram LM

- **\1-grams:**

```
-1.2041 <UNK>            0.0000
-1.2041 </s>             0.0000
-1.2041 <s> -0.2730
-0.4260 one -0.5283
-1.2041 three           -0.2730
-0.4260 two -0.5283
```

- **\2-grams:**

```
-0.1761 <s> one      0.0000
-0.4771 one three    0.1761
-0.3010 one two      0.3010
-0.1761 three two    0.0000
-0.3010 two one      0.3010
-0.4771 two three    0.1761
```

- **\3-grams:**

```
-0.3010 <s> one two
-0.3010 one three two
-0.4771 one two one
-0.4771 one two three
-0.3010 three two one
-0.4771 two one three
-0.4771 two one two
-0.3010 two three two
```

Step 1: Add explicit N-grams

- \1-grams:

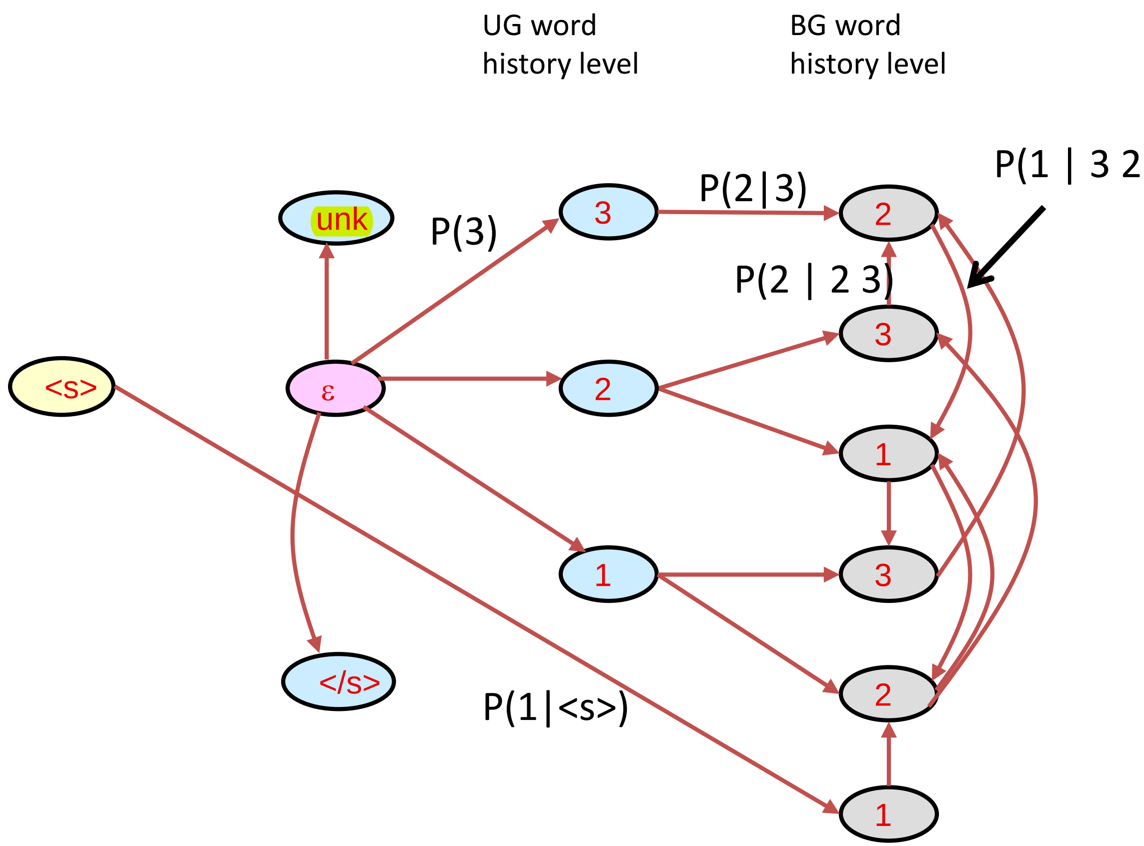
```
-1.2041 <UNK>             0.0000
-1.2041 </s>              0.0000
-1.2041 <s> -0.2730
-0.4260 one -0.5283
-1.2041 three            -0.2730
-0.4260 two -0.5283
```

- \2-grams:

```
-0.1761 <s> one       0.0000
-0.4771 one three     0.1761
-0.3010 one two       0.3010
-0.1761 three two     0.0000
-0.3010 two one       0.3010
-0.4771 two three     0.1761
```

- \3-grams:

```
-0.3010 <s> one two
-0.3010 one three two
-0.4771 one two one
-0.4771 one two three
-0.3010 three two one
-0.4771 two one three
-0.4771 two one two
-0.3010 two three two
```



Note: The two-word history out of every node in the bigram word history level is unique

- Note "EPSILON" Node for Unigram Probs

Step 2: Add backoffs

- **\1-grams:**

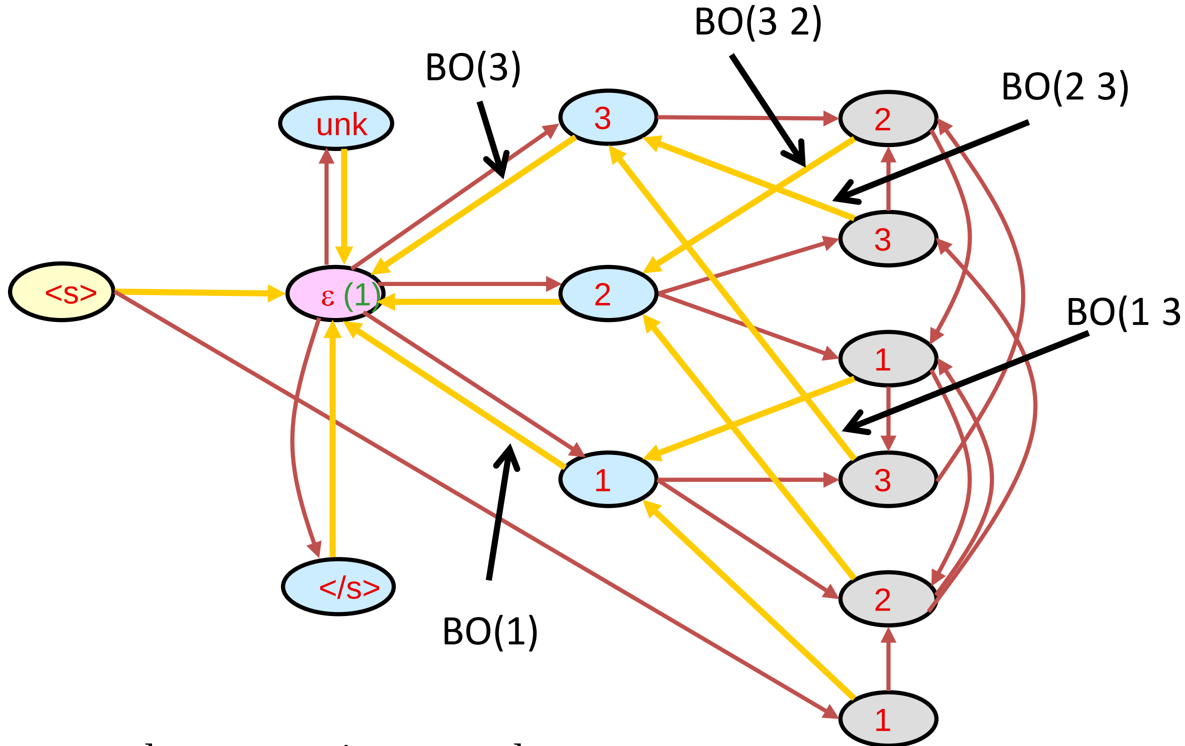
```
-1.2041 <UNK>          0.0000
-1.2041 </s>           0.0000
-1.2041 <s> -0.2730
-0.4260 one -0.5283
-1.2041 three         -0.2730
-0.4260 two -0.5283
```

- **\2-grams:**

```
-0.1761 <s> one       0.0000
-0.4771 one three     0.1761
-0.3010 one two       0.3010
-0.1761 three two     0.0000
-0.3010 two one       0.3010
-0.4771 two three     0.1761
```

- **\3-grams:**

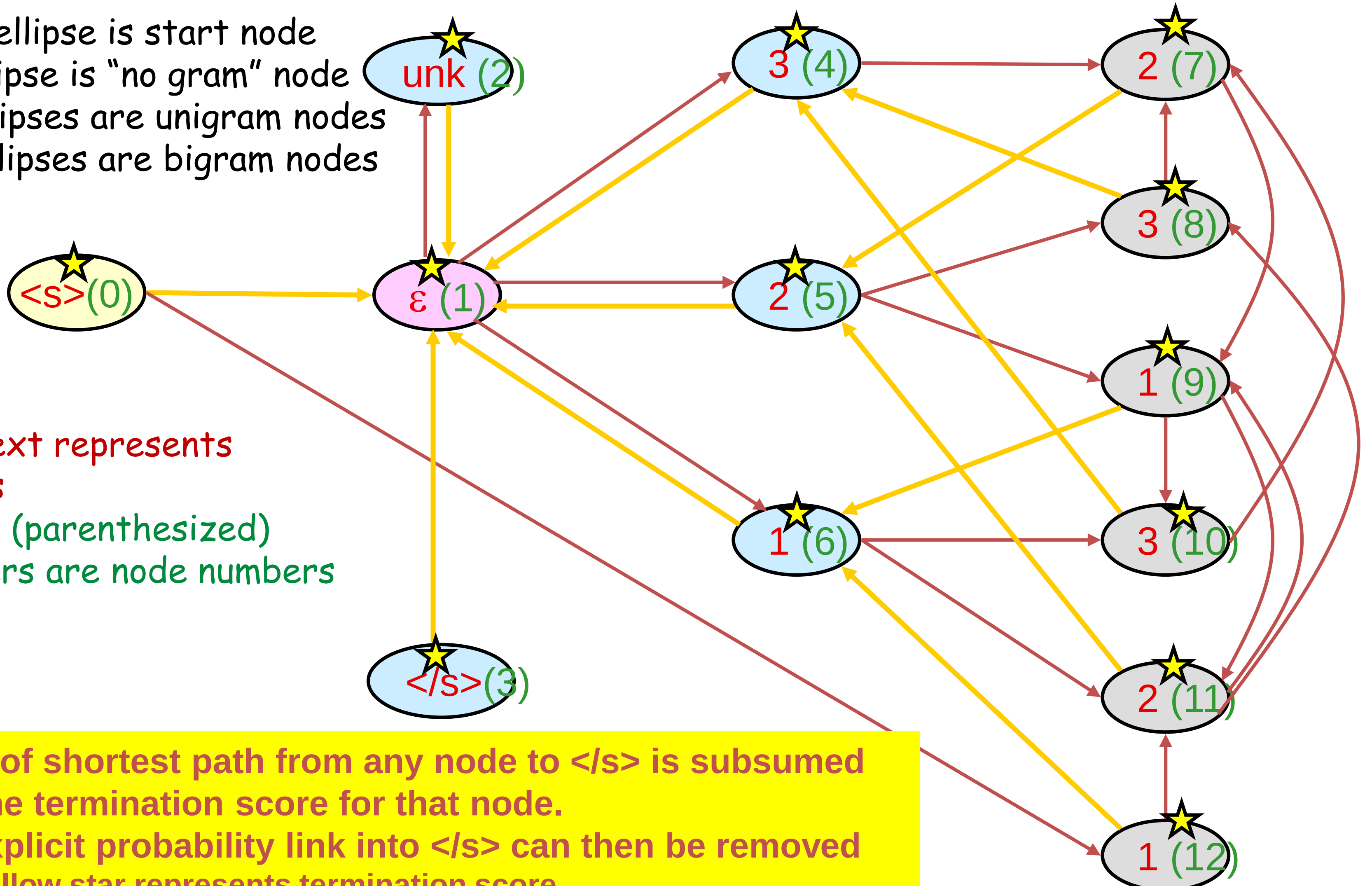
```
-0.3010 <s> one two
-0.3010 one three two
-0.4771 one two one
-0.4771 one two three
-0.3010 three two one
-0.4771 two one three
-0.4771 two one two
-0.3010 two three two
```



- From any node representing a word history "w_a" (unigram) add BO arc to epsilon
  - With score Backoff(wa)
- From any node representing a word history "w_a w_b" add a BO arc to w_b
  - With score Backoff (w_a w_b)

Ngram to FSG conversion: FSG

- Yellow ellipse is start node
- Pink ellipse is "no gram" node
- Blue ellipses are unigram nodes
- Gray ellipses are bigram nodes

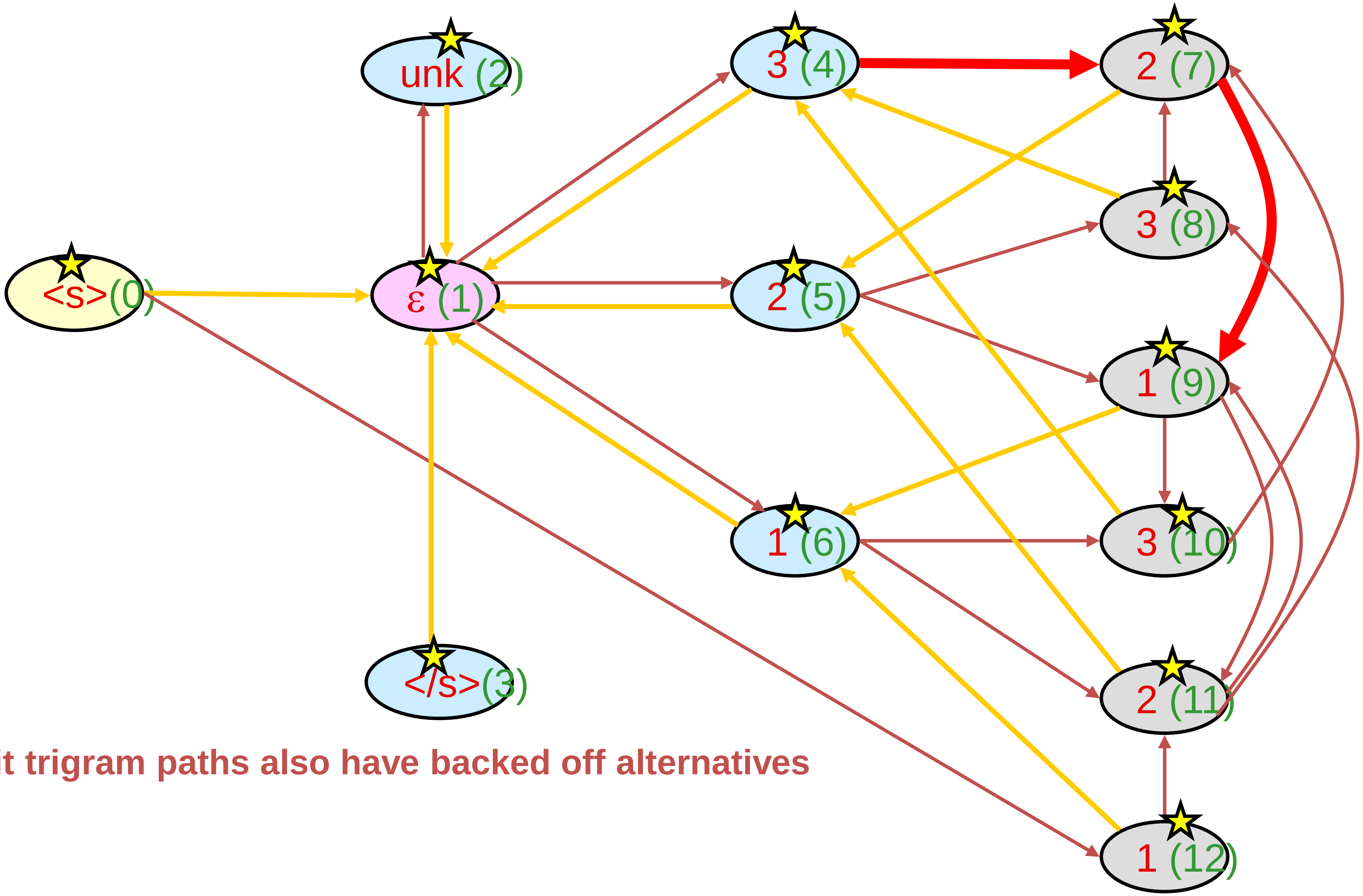


- red text represents words
- Green (parenthesized) numbers are node numbers

- Score of shortest path from any node to </s> is subsumed into the termination score for that node.
- The explicit probability link into </s> can then be removed
  - Yellow star represents termination score

A Problem: Paths are Duplicated

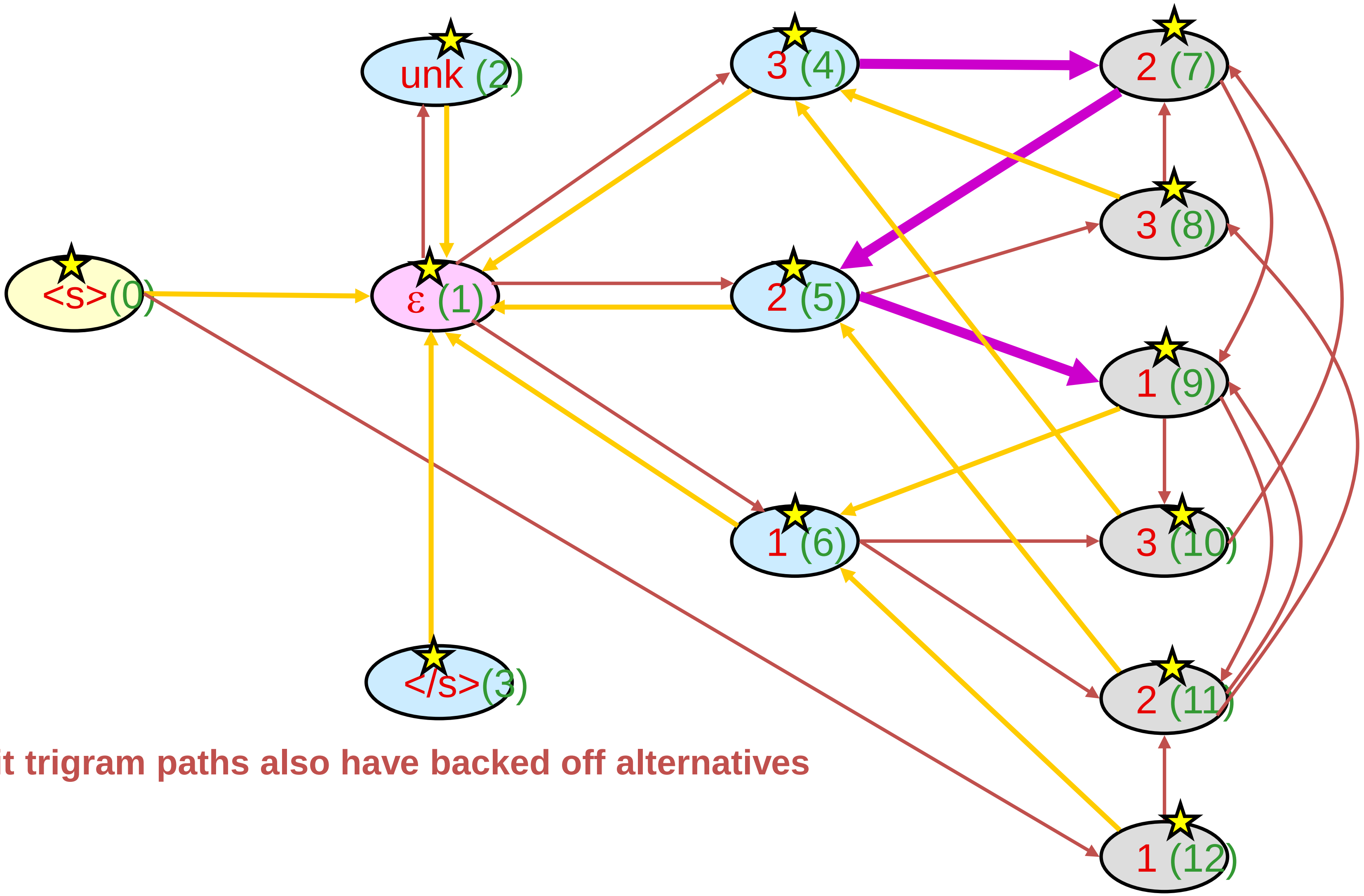
Explicit trigram path for trigram "three two one"



- Explicit trigram paths also have backed off alternatives

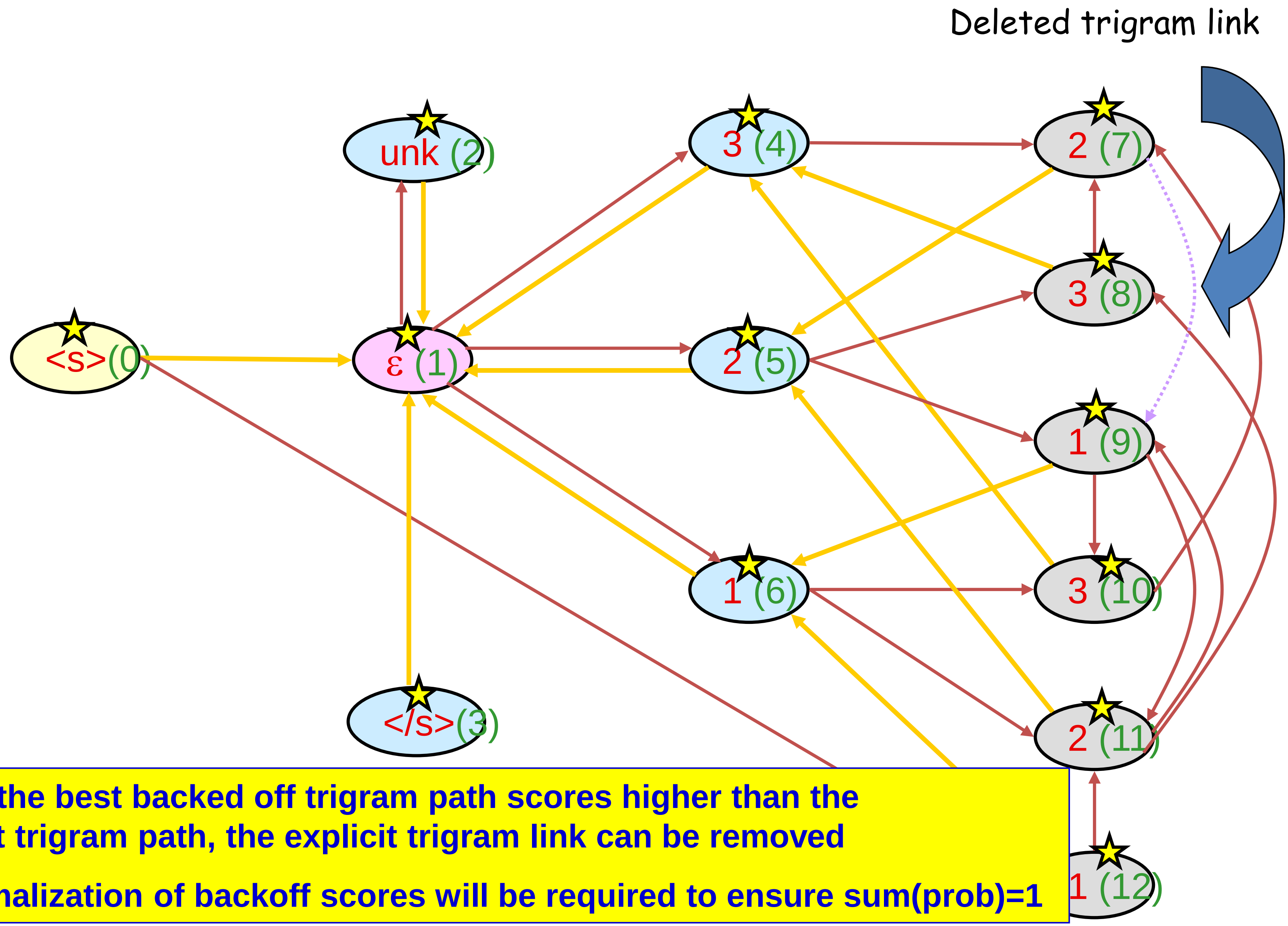
Backoff paths exist for explicit Ngrams

Backoff trigram path for trigram "three two one"



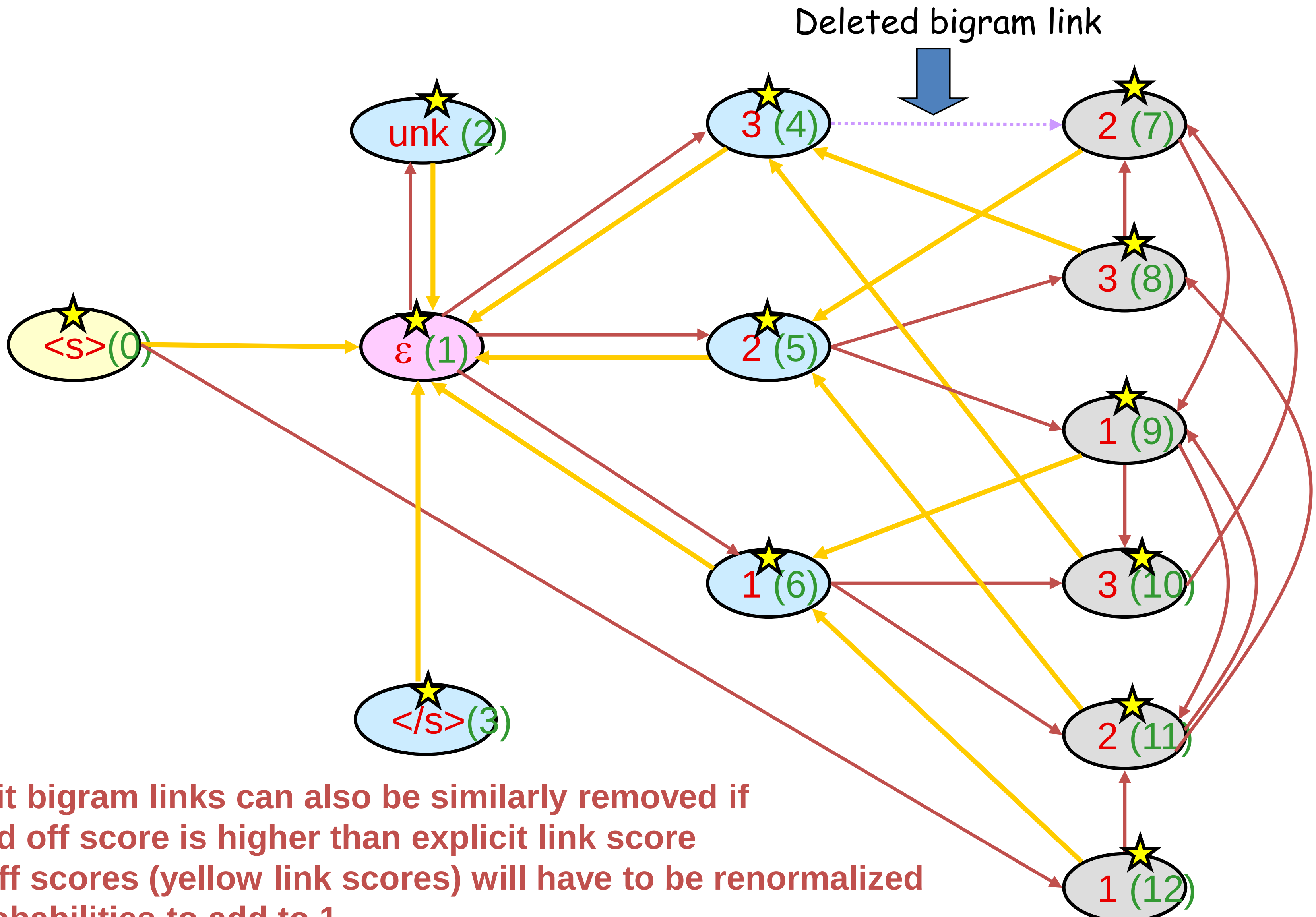
- **Explicit trigram paths also have backed off alternatives**

Delete “losing” edges



- When the best backed off trigram path scores higher than the explicit trigram path, the explicit trigram link can be removed
- Renormalization of backoff scores will be required to ensure sum(prob)=1

Delete “losing” edges



- **Explicit bigram links can also be similarly removed if backed off score is higher than explicit link score**
- **Backoff scores (yellow link scores) will have to be renormalized for probabilities to add to 1.**

Overall procedure for recognition with an Ngram language model

- Train HMMs for the acoustic model
- Train N-gram LM with backoff from training data
- Construct the Language graph, and from it the language HMM
  - Represent the Ngram language model structure as a compacted N-gram graph, as shown earlier
  - The graph must be dynamically constructed during recognition – it is usually too large to build statically
  - Probabilities on demand: Cannot explicitly store all K^N probabilities in the graph, and must be computed on the fly
    - K is the vocabulary size
  - Other, more compact structures, such as FSAs can also be used to represent the lanauge graph
    - later in the course
- Recognize