Automatic Speech Recognition in (just over) an Hour!

Class 22. 6 Nov 2009

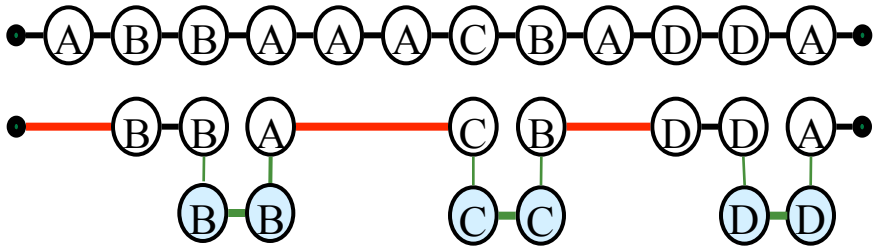
String Matching

- A simple problem: Given two strings of characters, how do we find the distance between them?
- Solution: Align them as best as we can, then measure the “cost” of aligning them

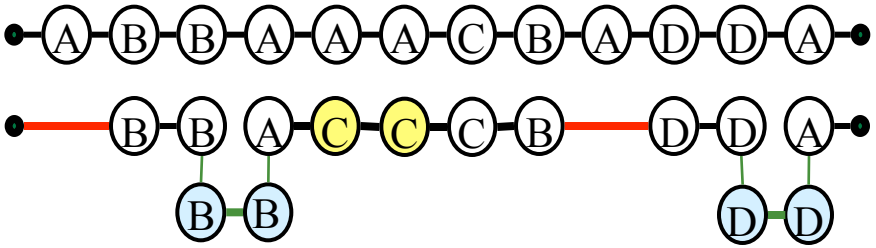
- Cost includes the costs of “insertion”, “Deletion”, “Substitution” and “Match”

Cost of match

- Match 1:
  - Insertions: B, B, C, C, D, D
  - Deletions: A, A, A, A
  - Matches: B, B, A, C, B, D, D, A
  - Total cost: $2I(C) + 2I(B) + 2I(D) + 4D(A) + 3M(B) + M(A) + M(C) + 2M(D)$



- Match 2:
  - Insertions: B, B, D, D
  - Deletions: A, A
  - Substitutions: (A,C), (A,C)
  - Matches: B, B, A, C, B, D, D, A
  - Total cost: $2I(B) + 2I(D) + 2D(A) + 2S(A,C) + 3M(B) + 2M(A) + M(C) + 2M(D)$



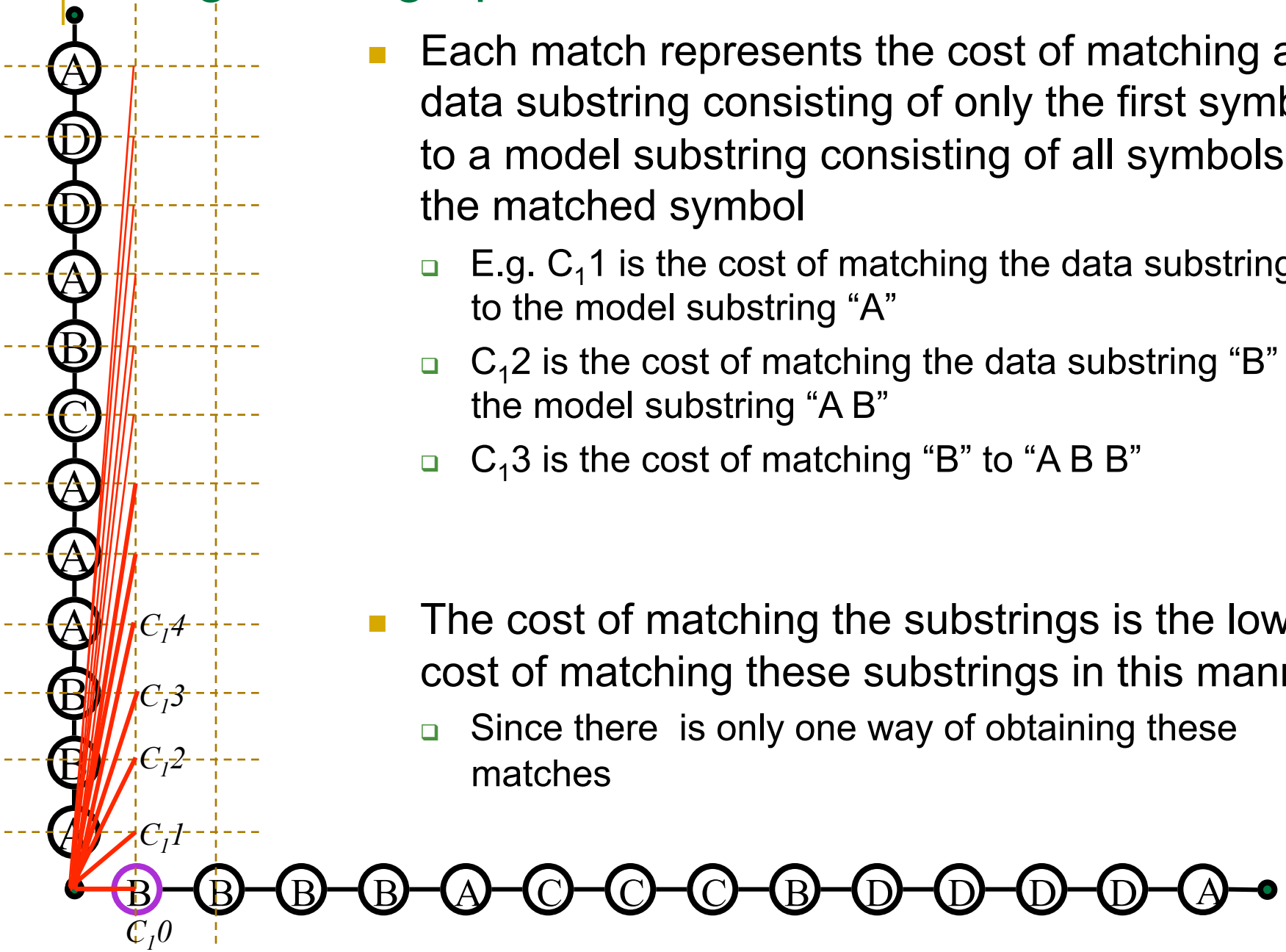
Computing the minimum cost

- The cost of matching a data string to a model string is the cost of the alignment that results in minimum cost
- How does one compute the lowest cost?
  - Exponentially large number of possibilities for matching two strings
  - Exhaustive evaluation of the cost of all possibilities to identify the minimum cost match is infeasible and unnecessary
  - The minimum cost can be efficiently computed using a dynamic programming algorithm that incrementally compares substrings of increasing length
    - Dynamic Time Warping

Dynamic Time Warping

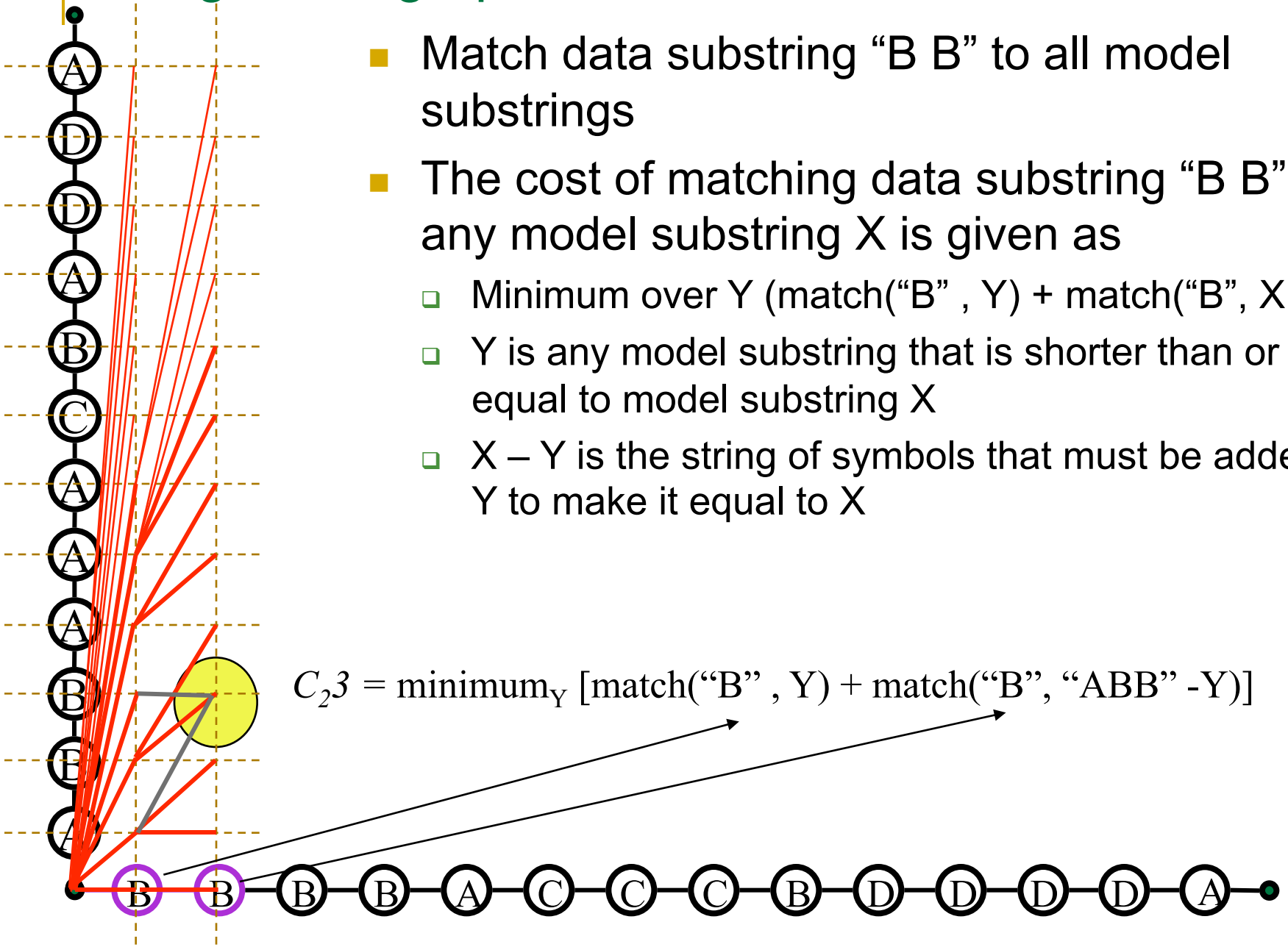
- Incrementally build up the best “alignment” by matching substrings to entire strings
- Standard procedure for edit distance: Computing the Levenshtein distance
  - Not possible to represent as a simple search through a static graph
    - Edge scores depend on symbols on the string..
- Alternative procedure – building and searching a static graph..

Alignment graph



- Each match represents the cost of matching a data substring consisting of only the first symbol, to a model substring consisting of all symbols until the matched symbol
  - E.g. $C_1 1$ is the cost of matching the data substring “B” to the model substring “A”
  - $C_1 2$ is the cost of matching the data substring “B” to the model substring “A B”
  - $C_1 3$ is the cost of matching “B” to “A B B”
- The cost of matching the substrings is the lowest cost of matching these substrings in this manner
  - Since there is only one way of obtaining these matches

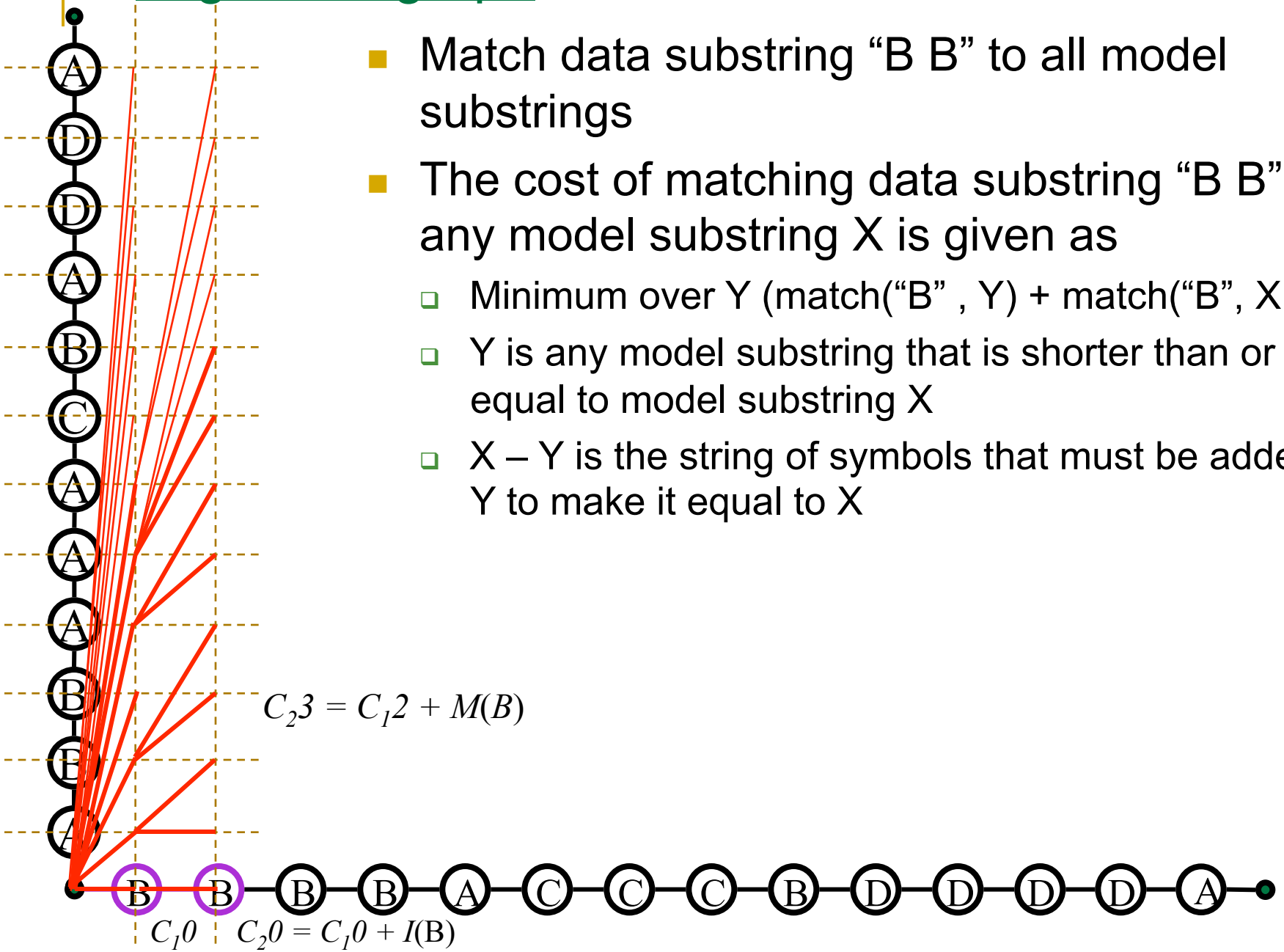
Alignment graph



- Match data substring “B B” to all model substrings
- The cost of matching data substring “B B” to any model substring X is given as
  - Minimum over Y (match(“B” , Y) + match(“B”, X -Y))
  - Y is any model substring that is shorter than or equal to model substring X
  - X – Y is the string of symbols that must be added to Y to make it equal to X

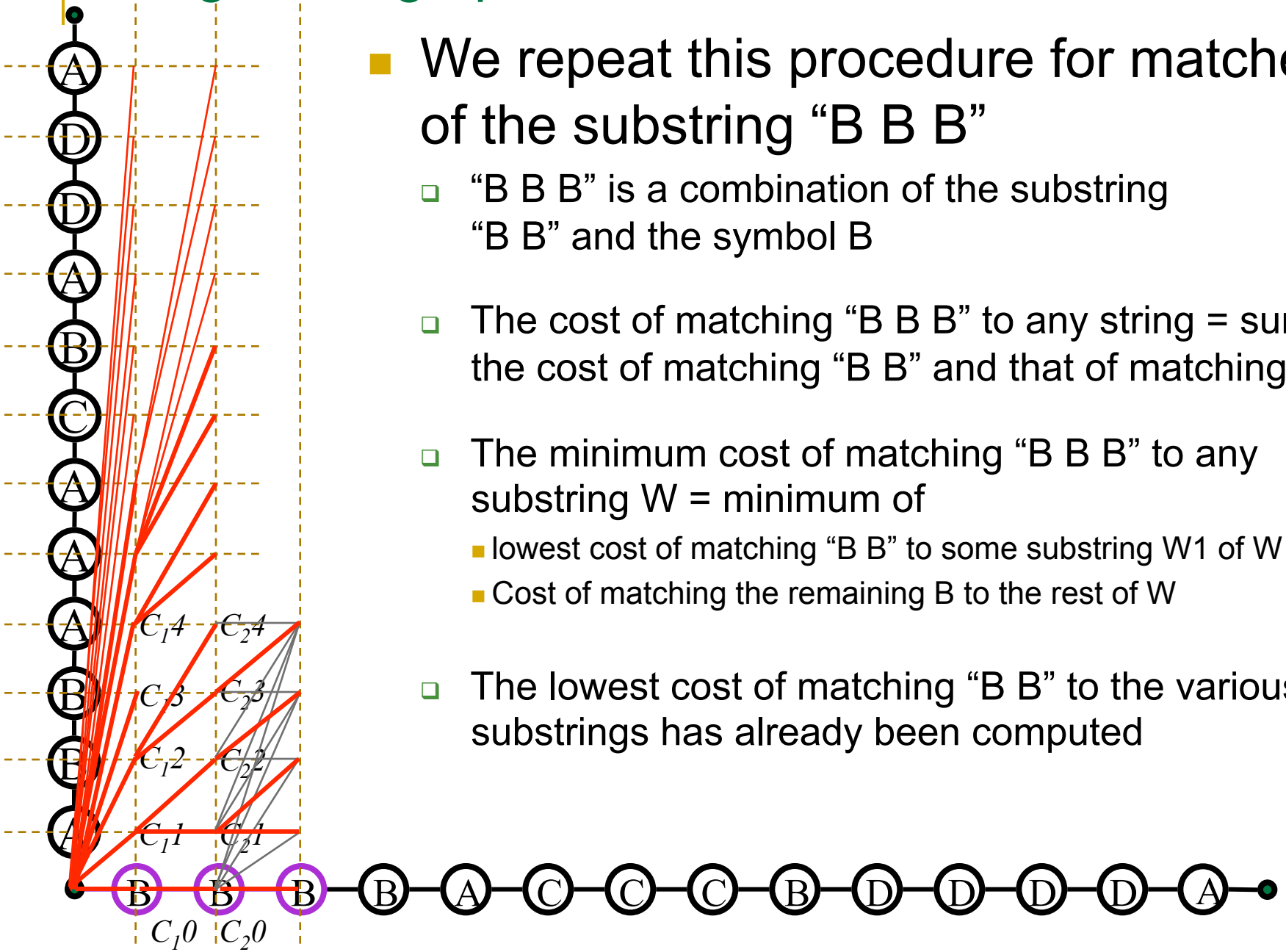
$C_2 3 = \text{minimum}_Y$ [match(“B” , Y) + match(“B”, “ABB” -Y)]

Alignment graph



- Match data substring “B B” to all model substrings
- The cost of matching data substring “B B” to any model substring X is given as
  - Minimum over Y (match(“B” , Y) + match(“B”, X -Y))
  - Y is any model substring that is shorter than or equal to model substring X
  - X – Y is the string of symbols that must be added to Y to make it equal to X

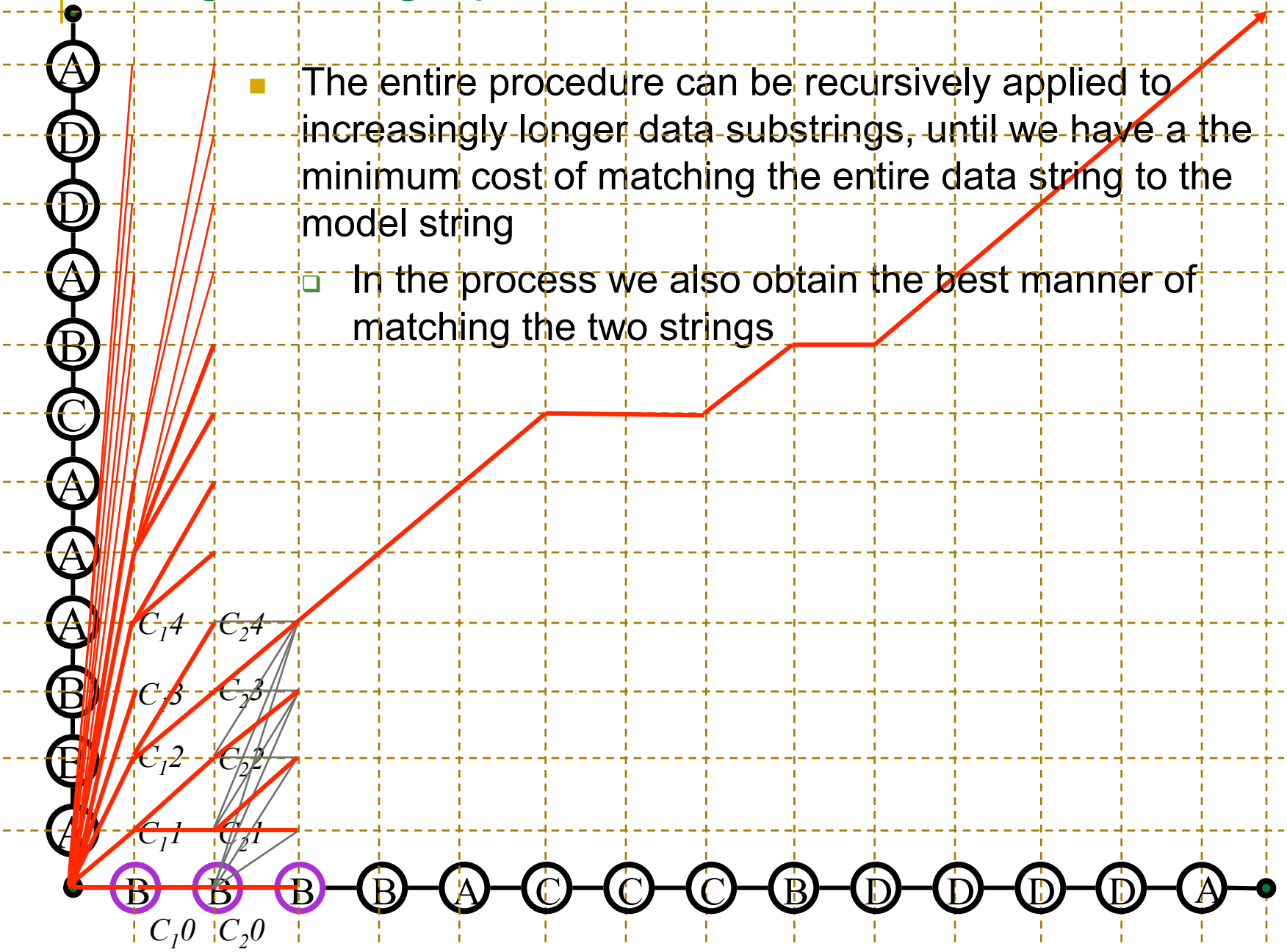
Alignment graph



- We repeat this procedure for matches of the substring “B B B”
  - “B B B” is a combination of the substring “B B” and the symbol B
  - The cost of matching “B B B” to any string = sum of the cost of matching “B B” and that of matching “B”
  - The minimum cost of matching “B B B” to any substring W = minimum of
    - lowest cost of matching “B B” to some substring W1 of W +
    - Cost of matching the remaining B to the rest of W
  - The lowest cost of matching “B B” to the various substrings has already been computed

Alignment graph

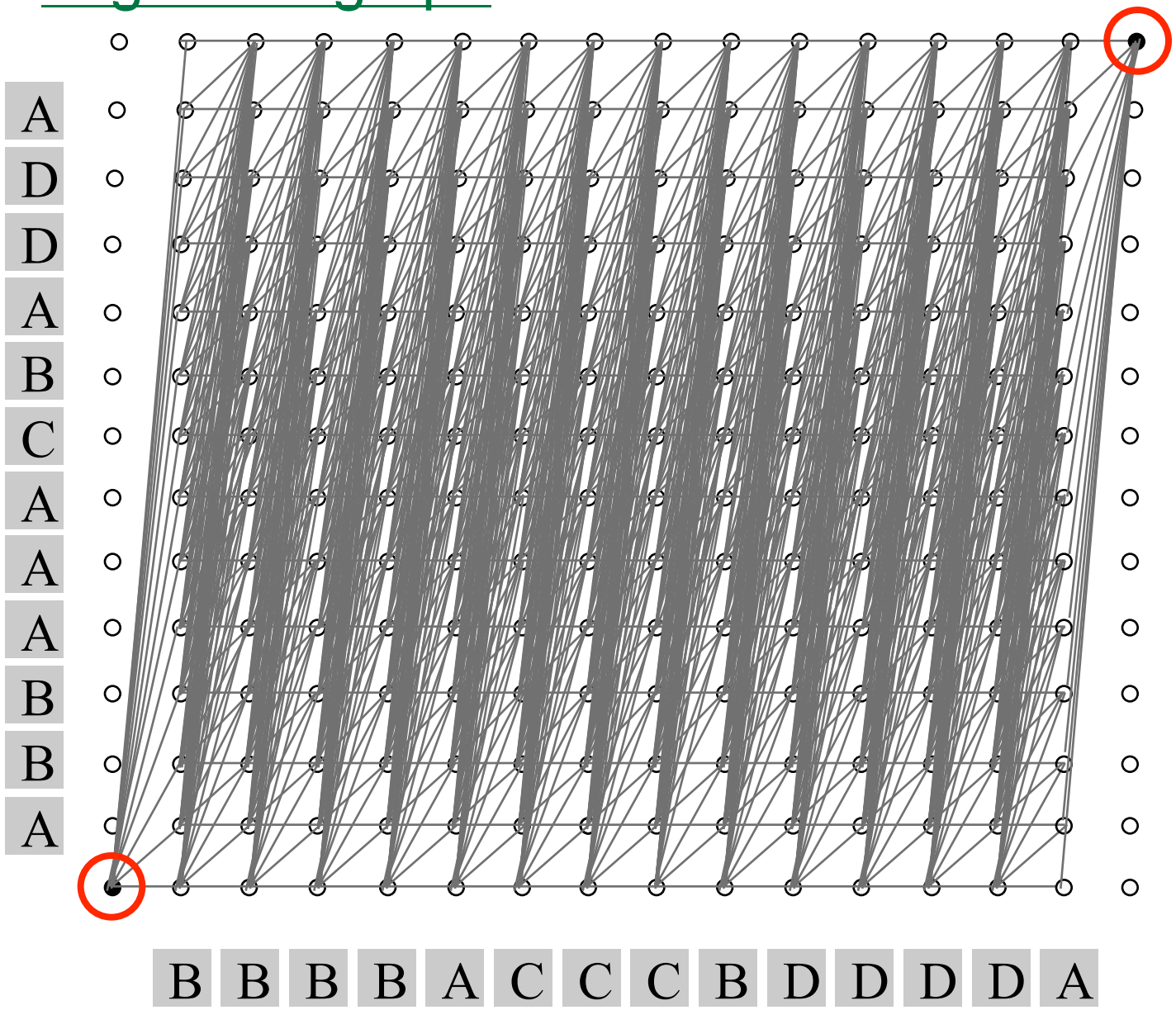
- The entire procedure can be recursively applied to increasingly longer data substrings, until we have a the minimum cost of matching the entire data string to the model string
  - In the process we also obtain the best manner of matching the two strings



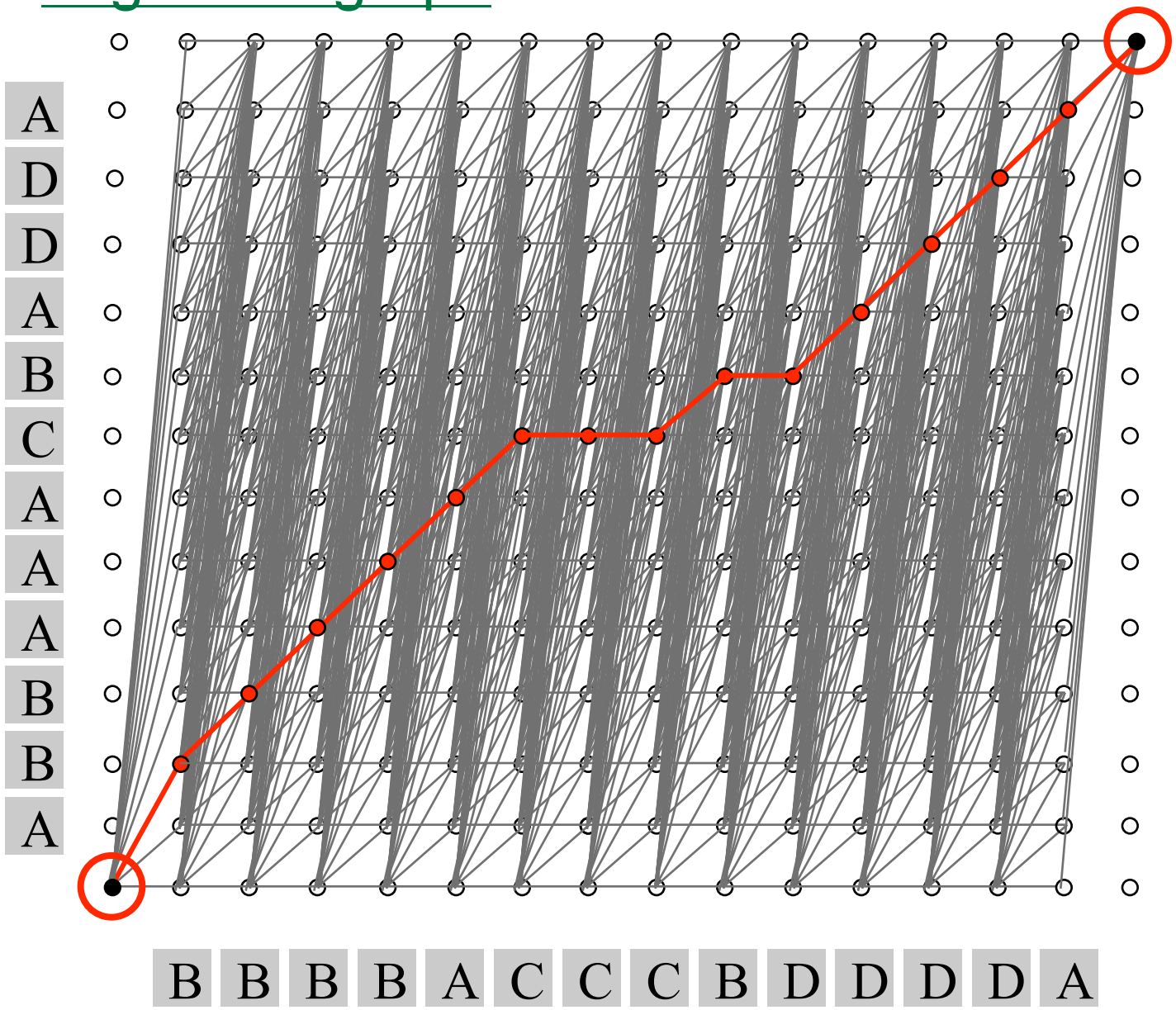
Aligning two strings

- The alignment process can be viewed as graph search

Alignment graph



Alignment graph



String matching

- This is just one way of creating the graph
  - The graph is asymmetric
    - Every symbol along the horizontal axis must be visited
    - Symbols on the vertical axis may be skipped
  - The resultant distance is not symmetric
    - Distance(string1, string2) != Distance(string2, string1)
- The graph may be constructed in other ways
  - Symmetric : symbols on horizontal axis may also be skipped
- Additional constraints may be incorporated
  - E.g. We may never delete more than one symbol in a sequence
  - Useful for the classification problems

Matching vector sequences

- The method is almost identical to what is done for string matching
- The crucial additional information is the notion of a distance between vectors
- The cost of substituting a vector A by a vector B is the distance between A and B
  - Distance could be computed using various metrics. E.g.
    - Euclidean distance is $\text{sqrt}(\Sigma_i |A_i - B_i|^2)$
    - Manhattan metric or the L1 norm: $\Sigma_i |A_i - B_i|$
    - Weighted Minkowski norms: $(\Sigma_i w_i |A_i - B_i|^n)^{1/n}$

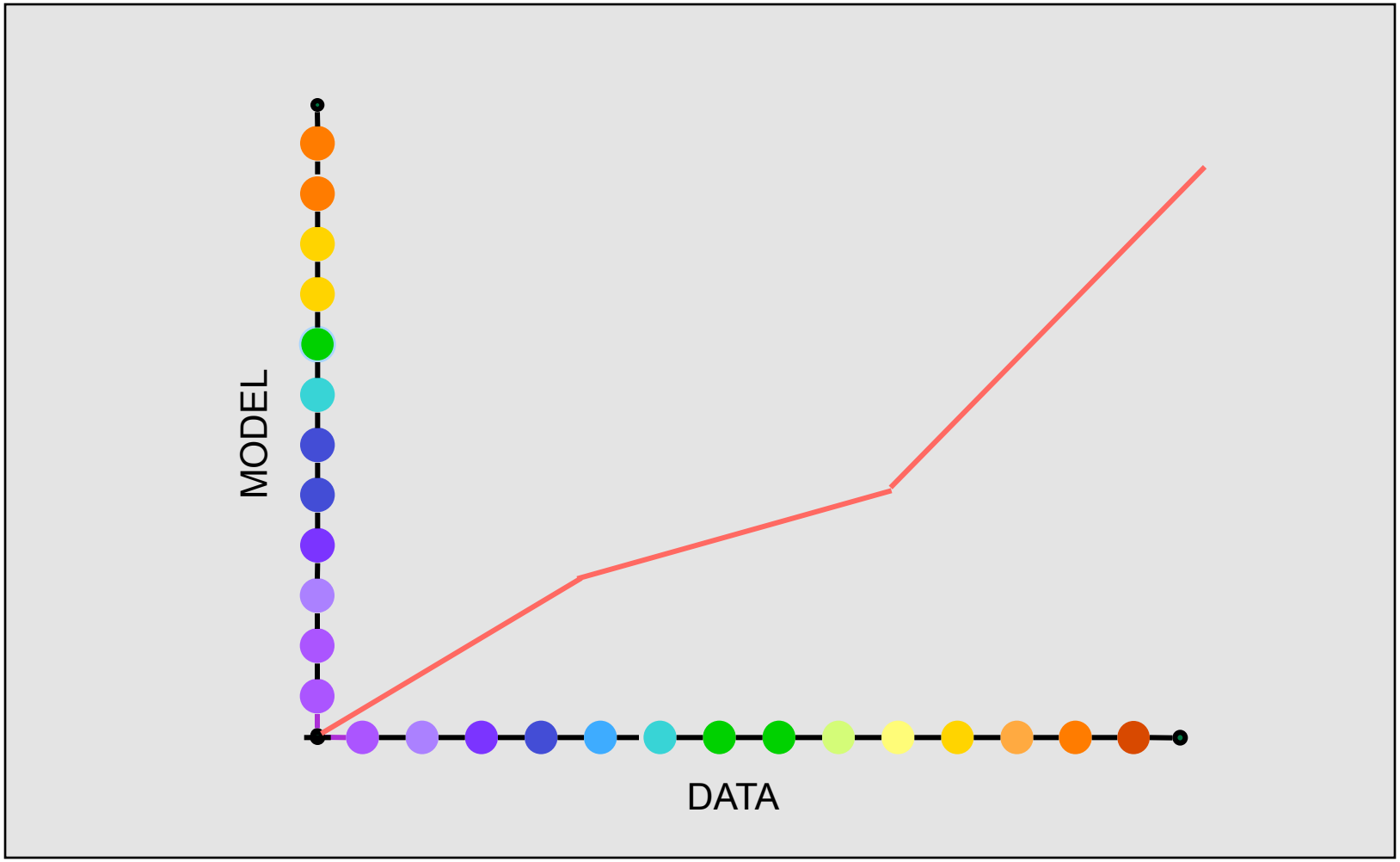
DTW and speech recognition

- Simple speech recognition (e.g. we want to recognize names for voice dialling)
- Store one or more examples of the speaker uttering each of the words as templates
- Given a new word, match the new recording against each of the templates
- Select the template for which the final DTW matching cost is lowest

Speech Recognition

- An “utterance” is actually converted to a sequence of cepstral vector prior to recognition
  - Both templates and new utterances
- Computing cepstra:
  - Window the signal into segments of 25ms, where adjacent segments overlap by 15ms
  - For each segment compute a magnitude spectrum
  - Compute the logarithm of the magnitude spectrum
  - Compute the Discrete Cosine Transform of the log magnitude spectrum
  - Retain only the first 13 components of the DCT
- Each utterance is finally converted to a sequence of 13-dimensional vectors
  - Optionally augmented by delta and double delta features
    - Potentially, with other processing such as mean and variance normalization
- Returning to our discussion...

DTW with two sequences of vectors

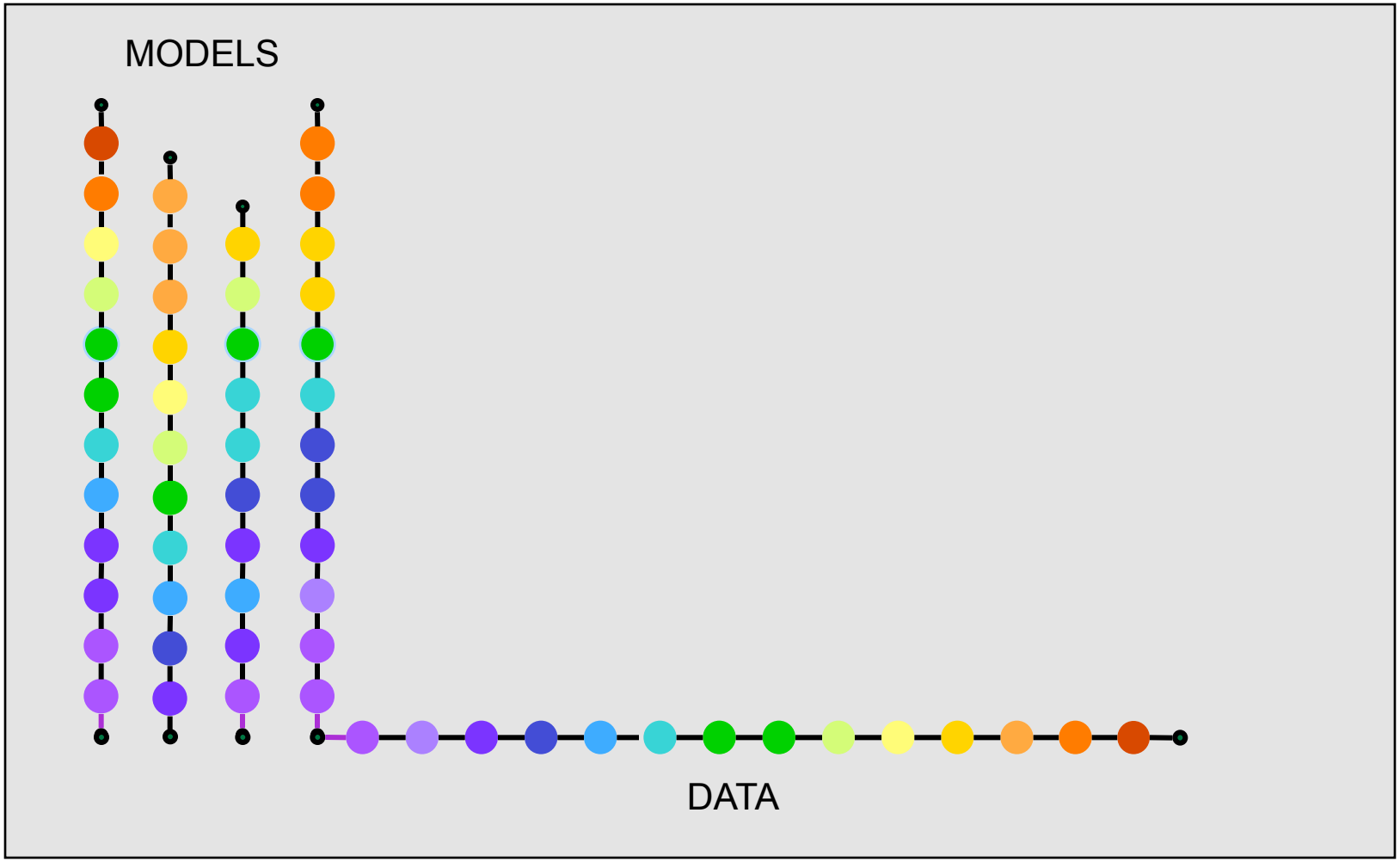


The template (model) is matched against the data string to be recognized
Select the template with the lowest cost of match

Using Multiple Templates

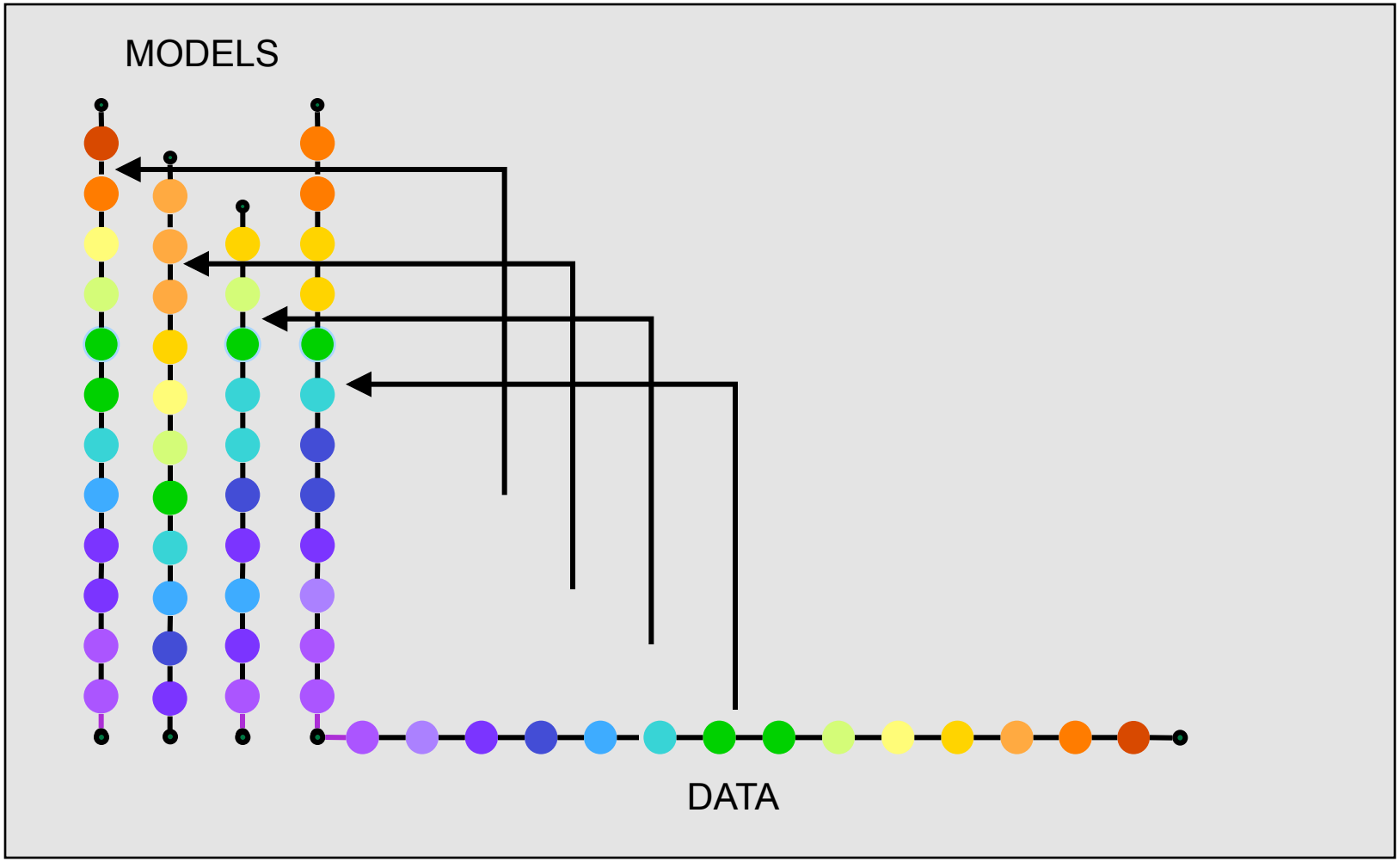
- A person may utter a word (e.g. ZERO) in multiple ways
  - In fact, one never utters the word twice in exactly the same way
- Store *multiple* templates for each word
  - Record 5 instances of “ZERO”, five of “ONE” etc.
- Recognition: Cost of word = cost of closest template of word (to test utterance)
  - Select minimum cost word as recognition output

DTW with multiple models



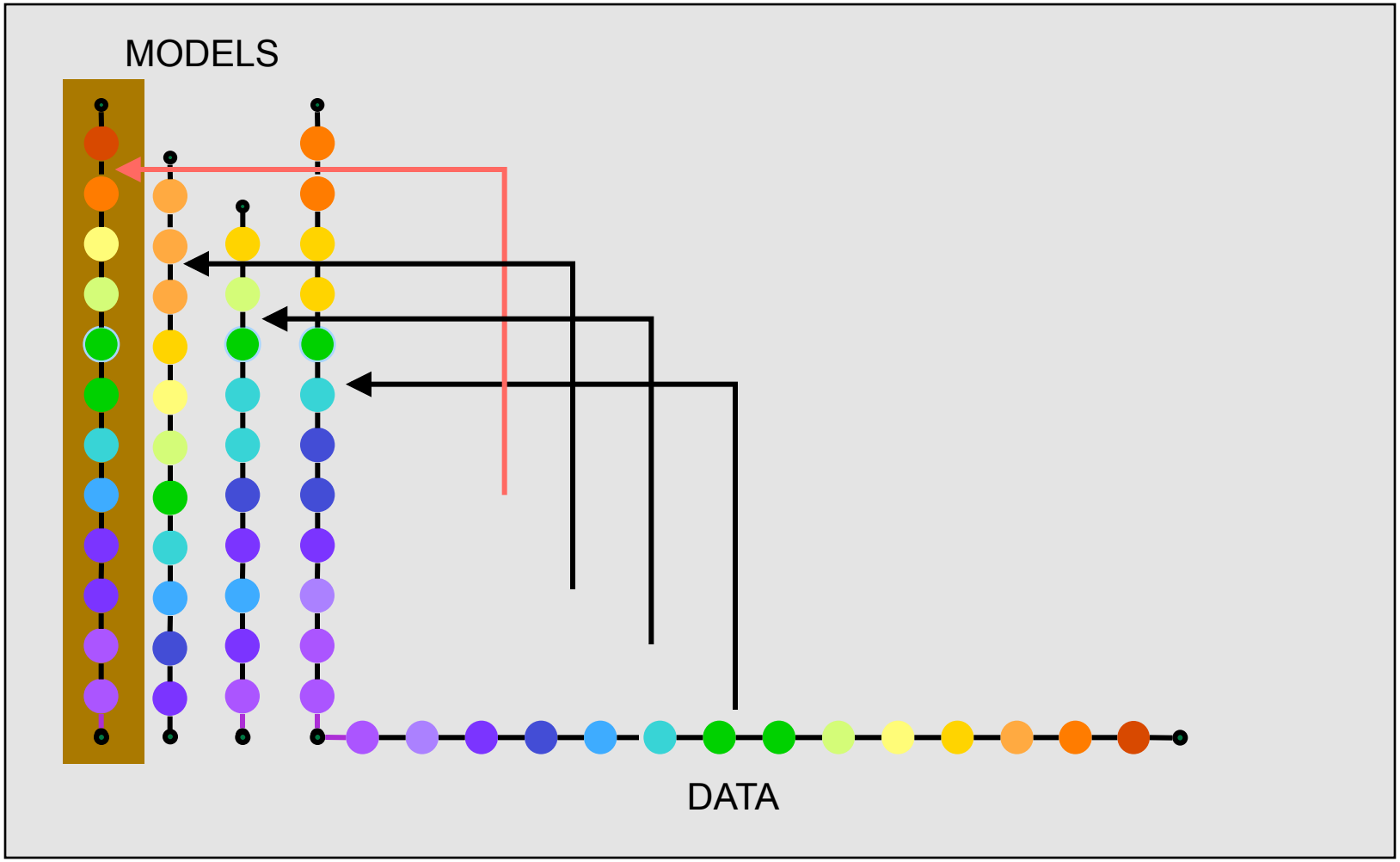
Evaluate all templates for a word against the data

DTW with multiple models



Evaluate all templates for a word against the data

DTW with multiple models

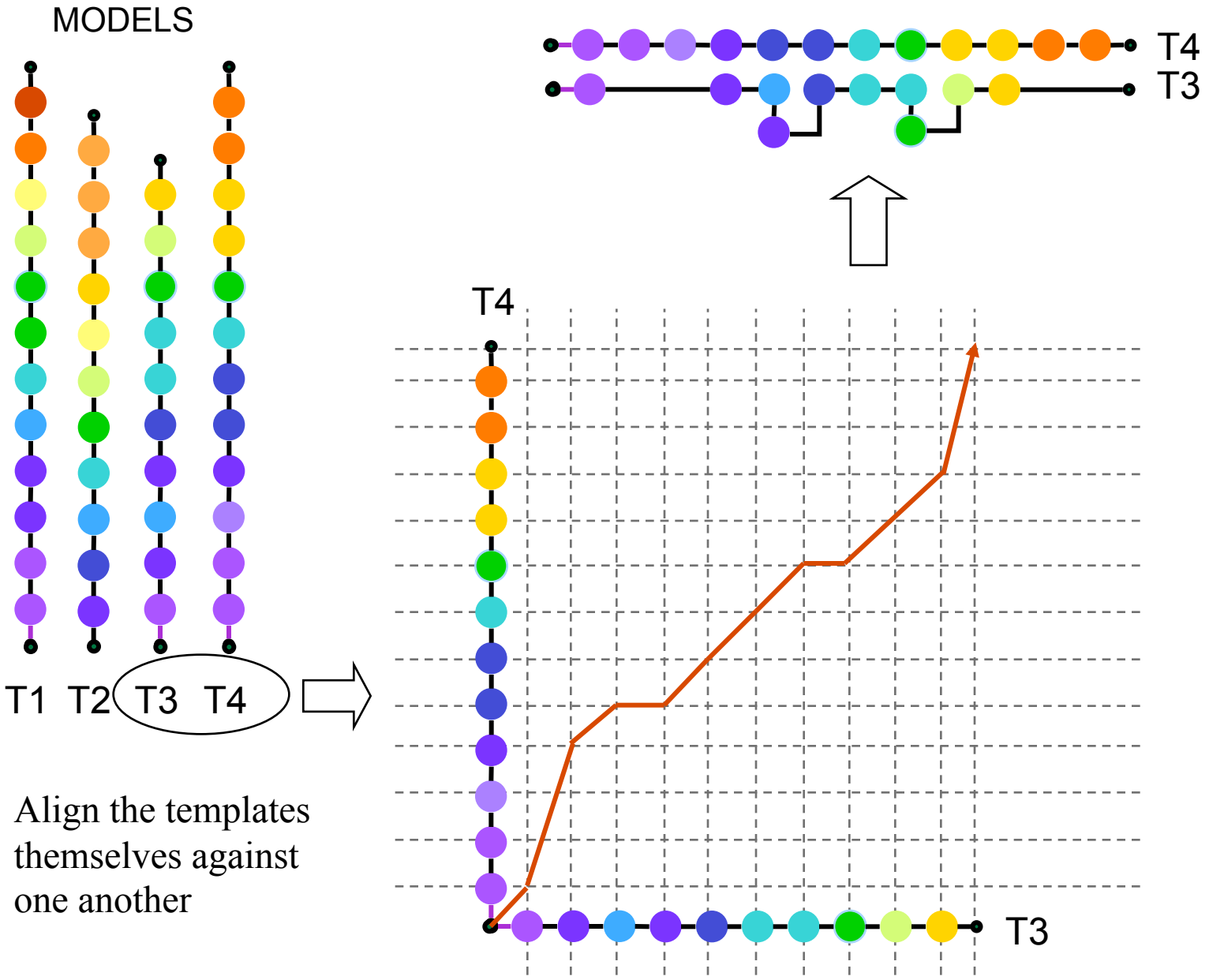


Evaluate all templates for a word against the data
Select the best fitting template. The corresponding cost is the cost of the match

The Problem with Multiple Templates

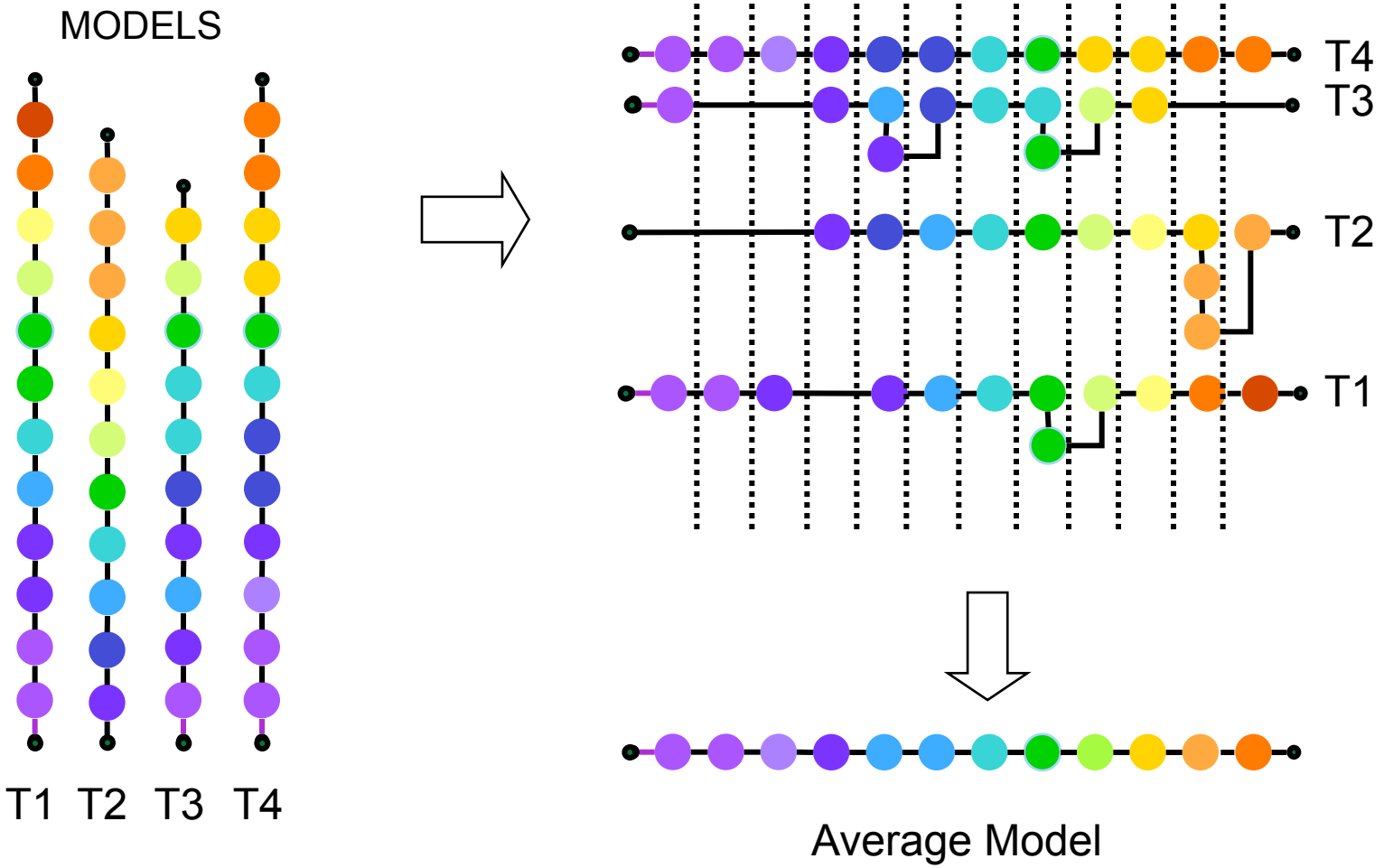
- Finding the closest template to a test utterance requires evaluation of all test templates
  - This is expensive
- Additionally, the set of templates may not cover all possible variants of the words
  - Must generalize from templates to represent other variants
- We do this by averaging the templates

DTW with multiple models



Align the templates themselves against one another

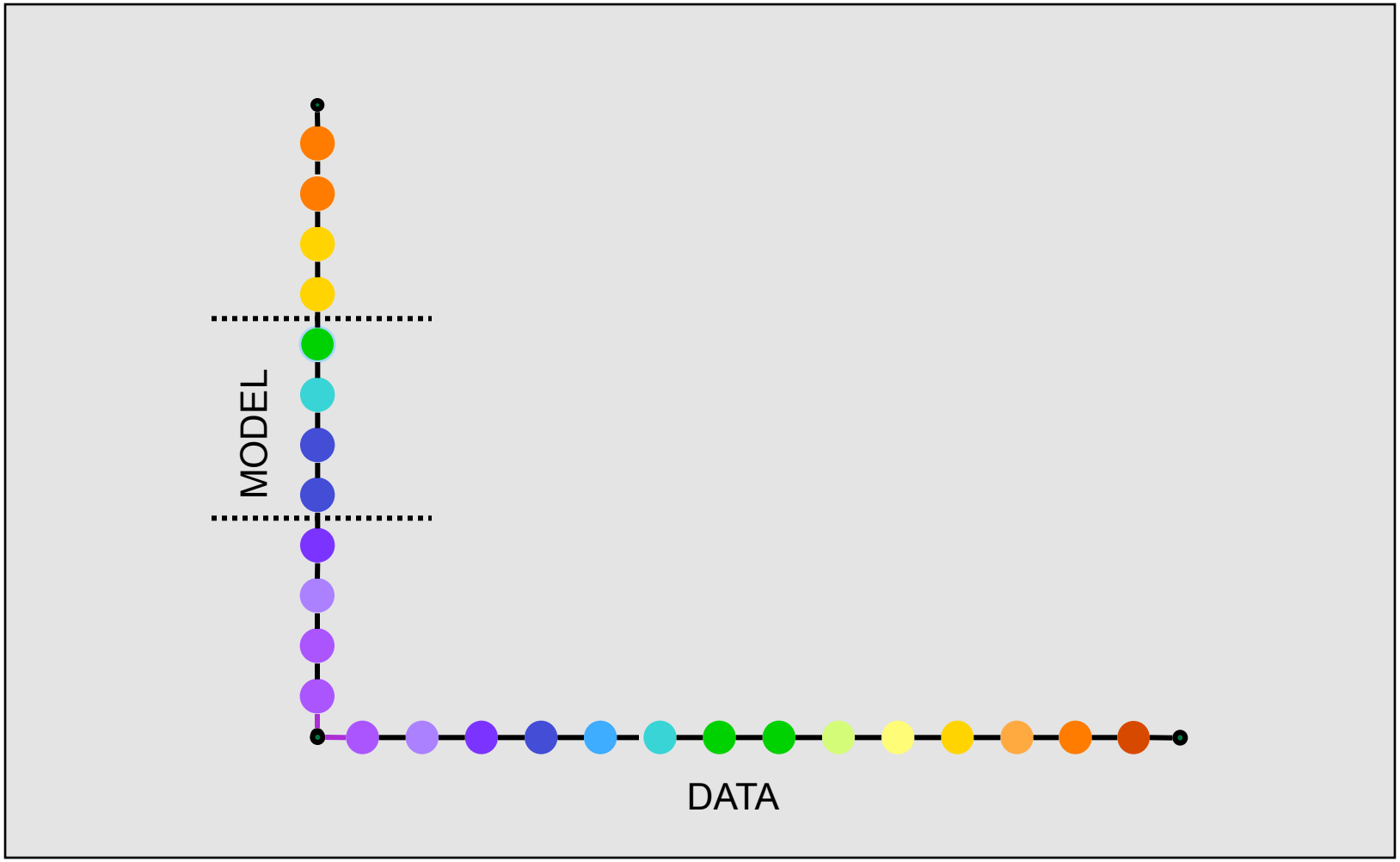
DTW with multiple models



Align the templates themselves against one another

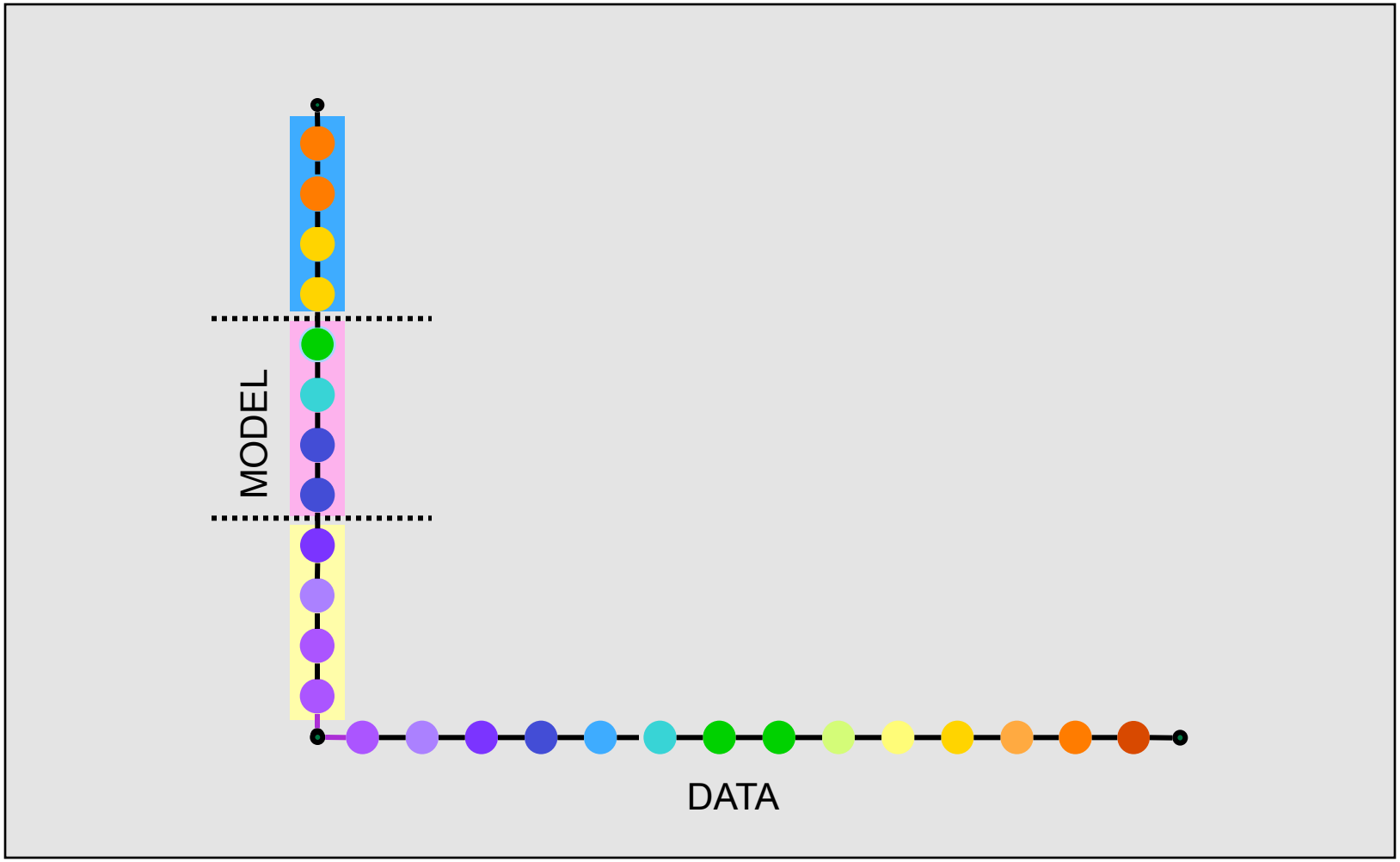
Average the aligned templates

DTW with one model



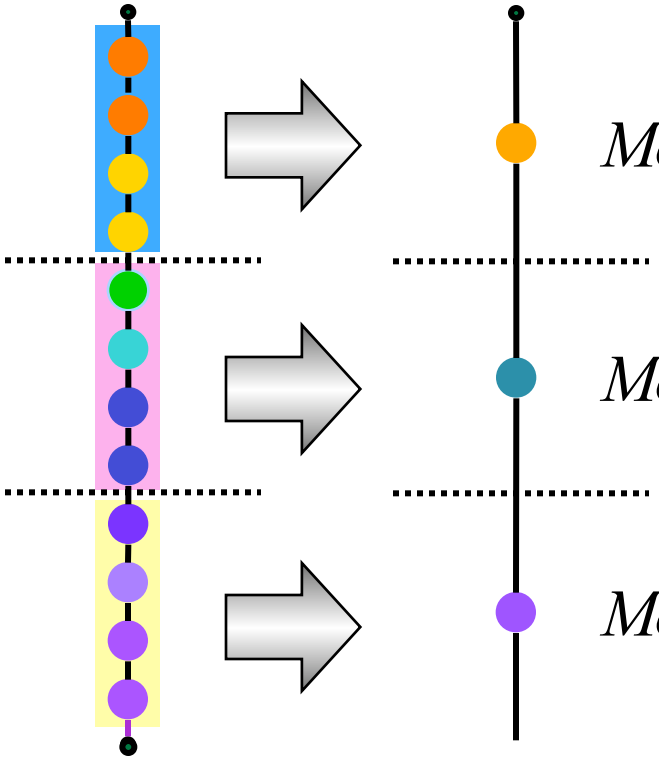
A SIMPLER METHOD: Segment the templates themselves and average within segments

DTW with one model



A simple trick: segment the “model” into regions of equal length
Average each segment into a single point

DTW with one model



$$Model\ Vector = \frac{1}{N} \sum_{i \in segment} vector(i)$$

$$Model\ Vector = \frac{1}{N} \sum_{i \in segment} vector(i)$$

$$Model\ Vector = \frac{1}{N} \sum_{i \in segment} vector(i)$$

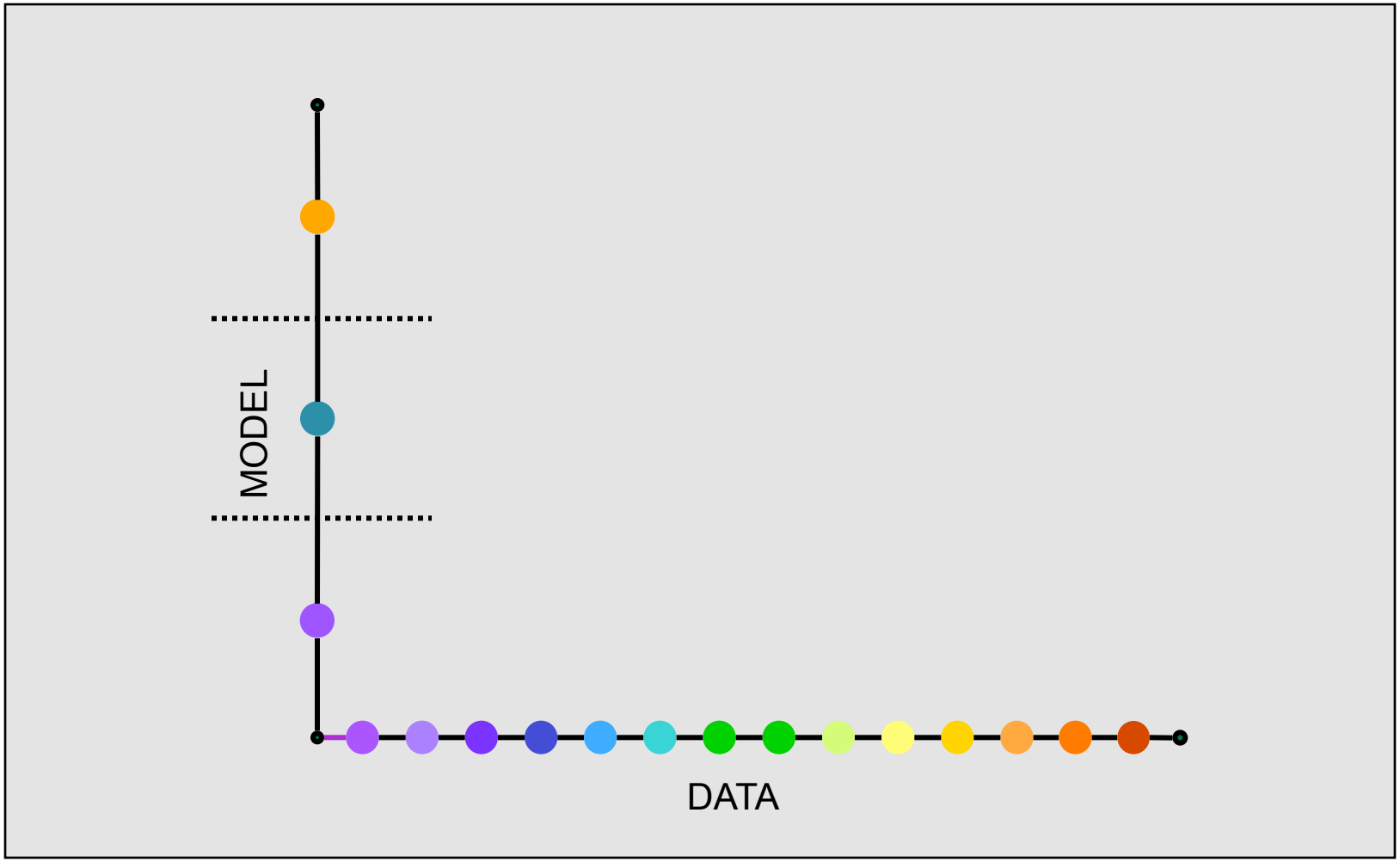
$$m_j = \frac{1}{N_j} \sum_{i \in segment(j)} v(i)$$

m_j is the model vector for the j^{th} segment

N_j is the number of training vectors in the j^{th} segment

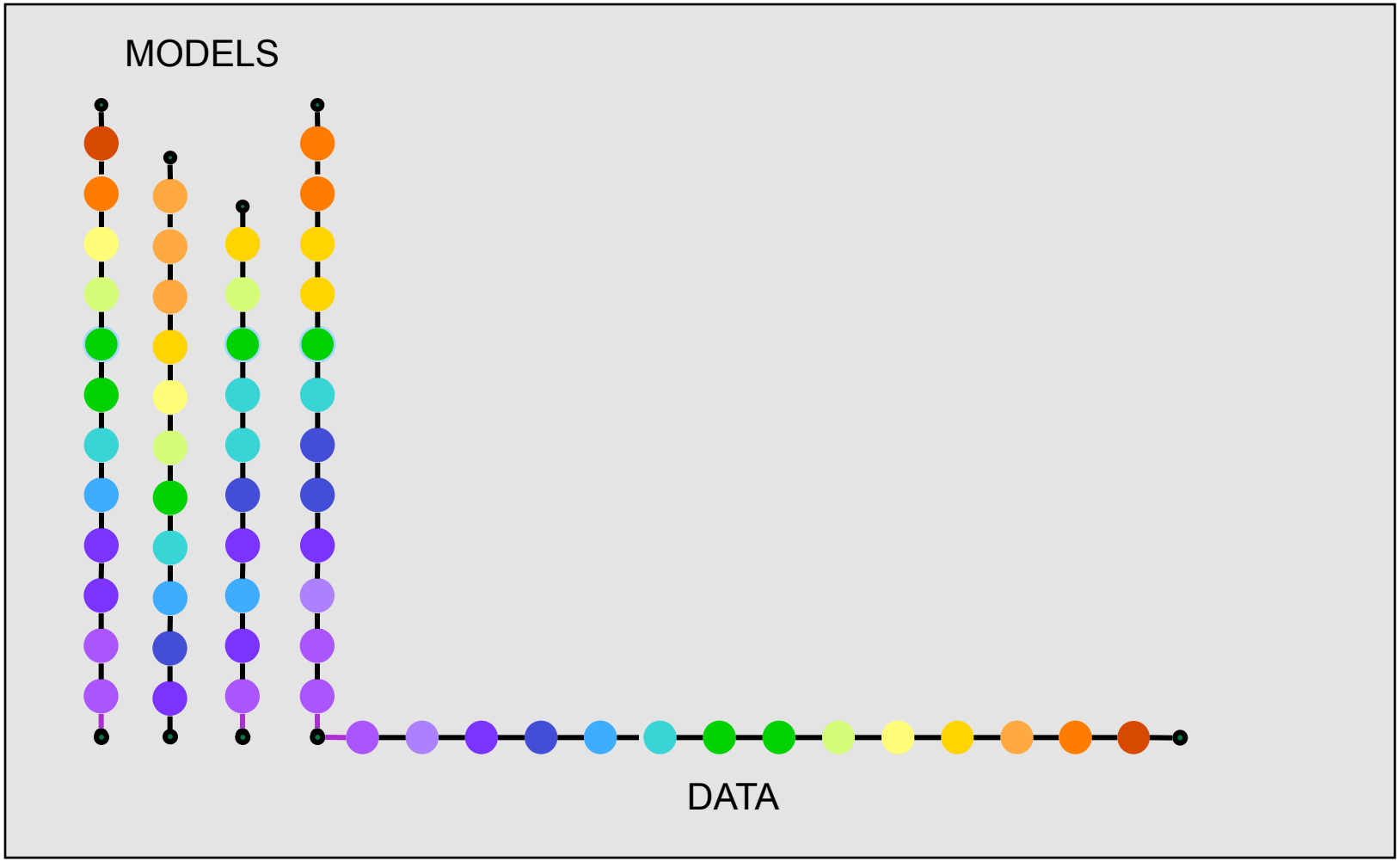
$v(i)$ is the i^{th} training vector

DTW with one model



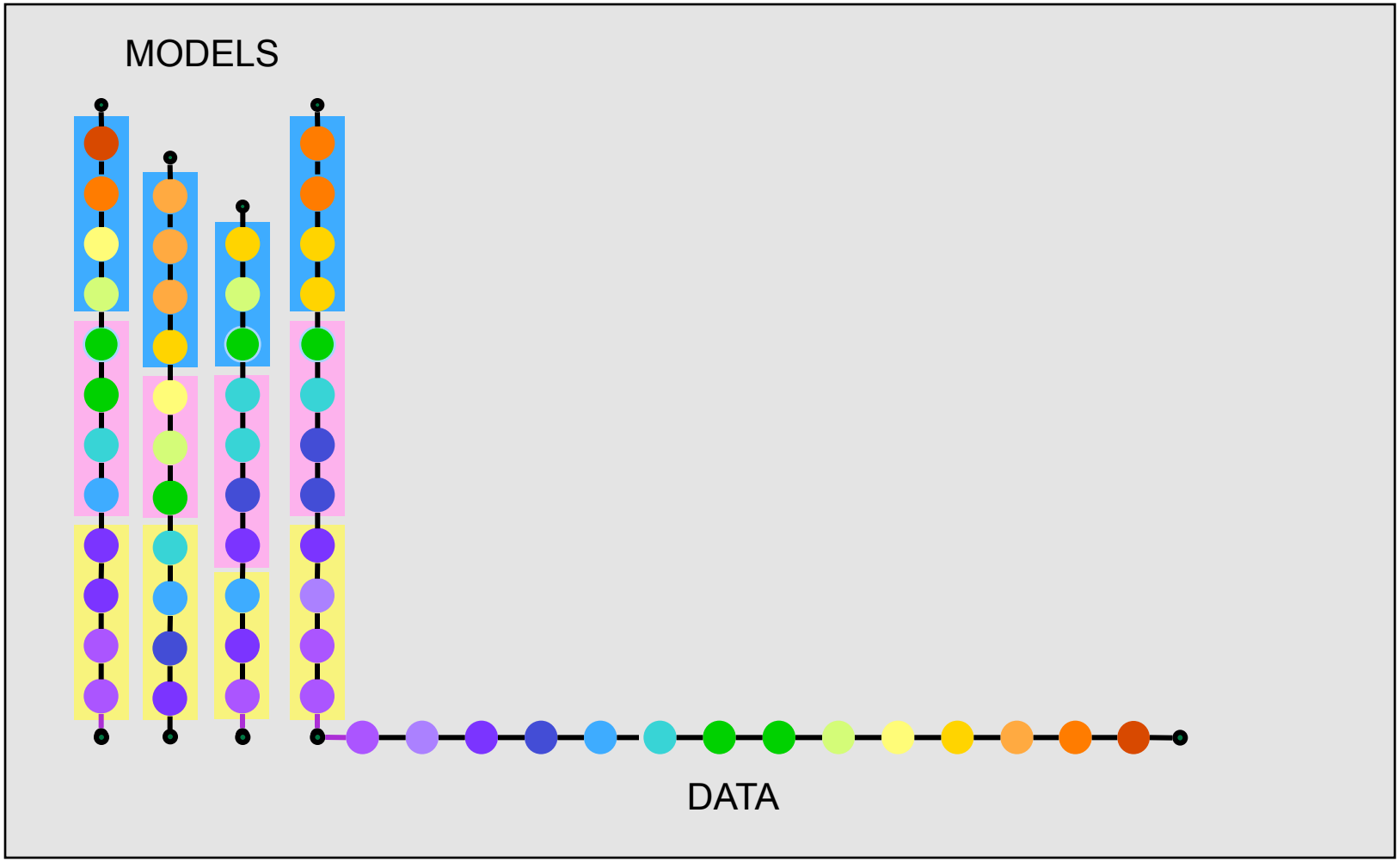
The averaged template is matched against the data string to be recognized
Select the word whose averaed template has the lowest cost of match

DTW with multiple models



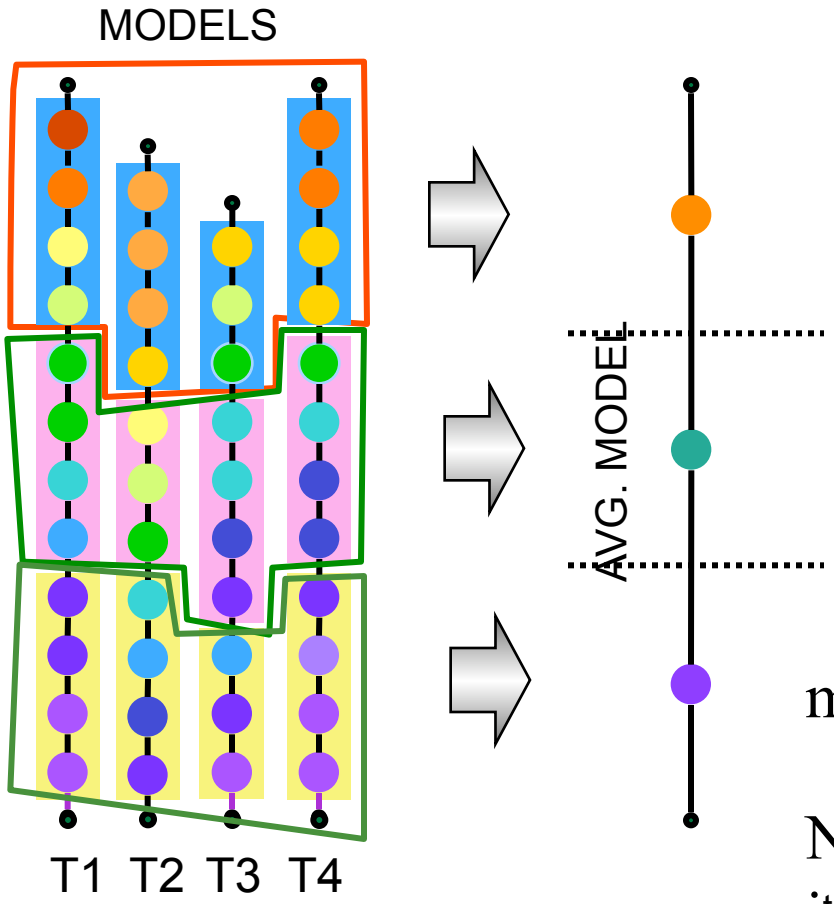
Segment all templates
Average each region into a single point

DTW with multiple models



Segment all templates
Average each region into a single point

DTW with multiple models



$$m_j = \frac{1}{\sum_k N_{k,j}} \sum_{i \in segment_k(j)} v_k(i)$$

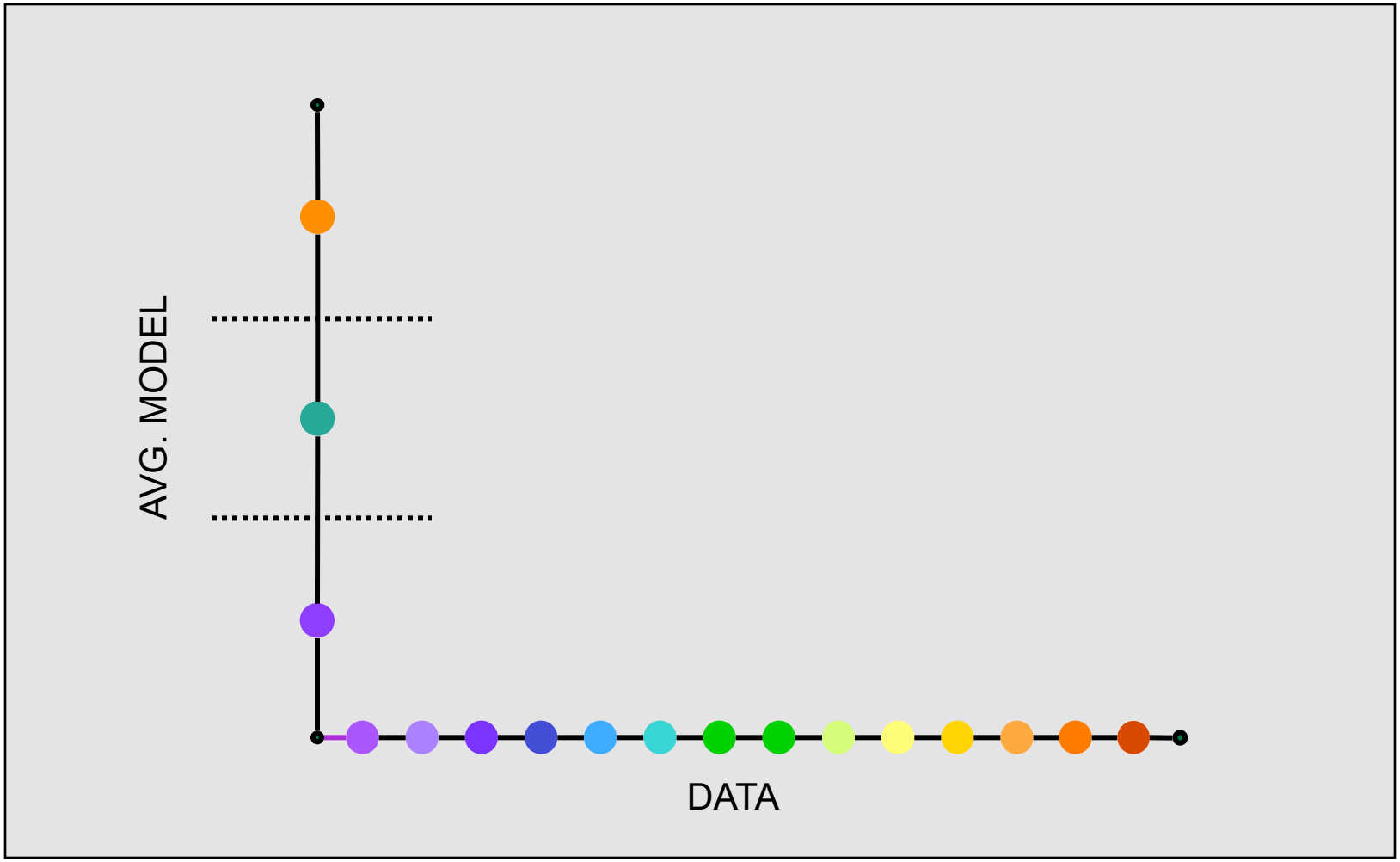
$segment_k(j)$ is the j^{th} segment of the k^{th} training sequence

m_j is the model vector for the j^{th} segment

$N_{k,j}$ is the number of training vectors in the j^{th} segment of the k^{th} training sequence

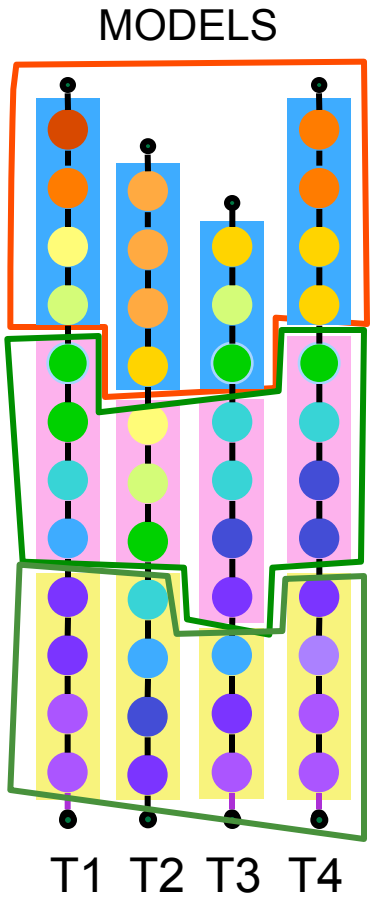
$v_k(i)$ is the i^{th} vector of the k^{th} training sequence

DTW with multiple models



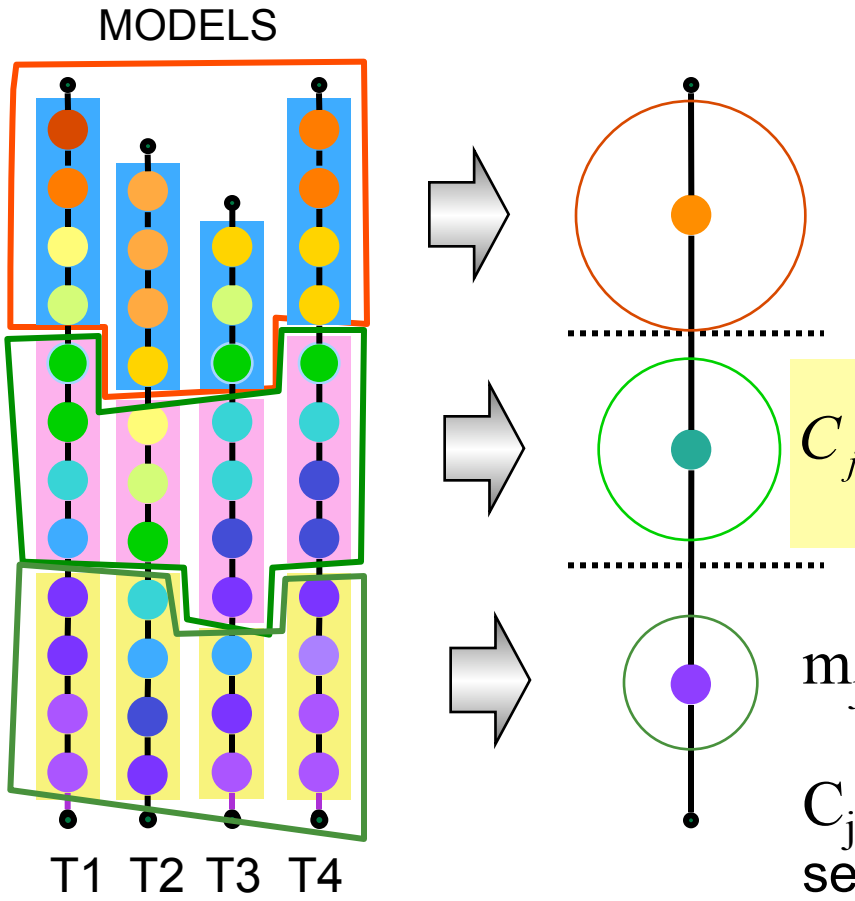
Segment all templates
Average each region into a single point
To get a simple average model, which is used for recognition

DTW with multiple models



- The inherent variation between vectors is different for the different segments
  - E.g. the variation in the colors of the beads in the top segment is greater than that in the bottom segment
- Ideally we should account for the differences in variation in the segments
  - E.g, a vector in a test sequence may actually be more matched to the central segment, which permits greater variation, although it is closer, in a Euclidean sense, to the mean of the lower segment, which permits lesser variation

DTW with multiple models



We can define the covariance for each segment using the standard formula for covariance

$$C_j = \frac{1}{\sum_k N_{k,j}} \sum_{i \in segment_k(j)} \left(v_k(i) - m_j\right)\left(v_k(i) - m_j\right)^T$$

m_j is the model vector for the j^{th} segment

C_j is the covariance of the vectors in the j^{th} segment

DTW with multiple models

- The distance function must be modified to account for the covariance
- Mahalanobis distance:
  - Normalizes contribution of all dimensions of the data

$$d(v, m_j) = (v - m_j)^T C_j^{-1} (v - m_j)$$

  - v is a data vector, m_j is the mean of a segment, C_j is the covariance matrix for the segment

- Negative Gaussian log likelihood:
  - Assumes a Gaussian distribution for the segment and computes the probability of the vector on this distribution

$$Gaussian(v; m_j, C_j) = \frac{1}{\sqrt{2\pi |C_j|}} e^{-0.5(v-m_j)^T C_j^{-1} (v-m_j)}$$

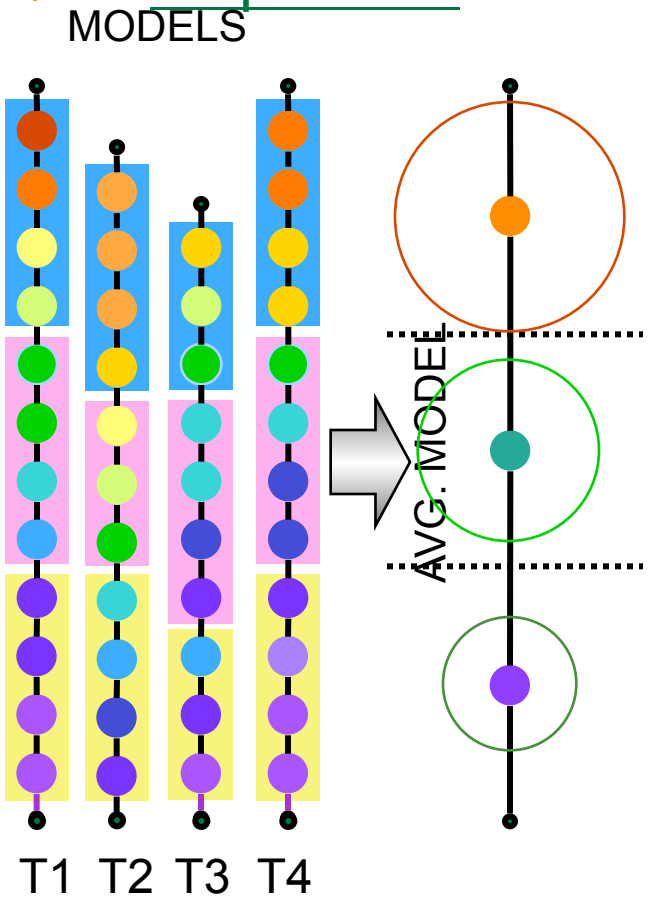
$$d(v, m_j) = -\log\left(Gaussian(v; m_j, C_j)\right)$$

$$= -0.5 \log\left(2\pi |C_j|\right) + 0.5(v - m_j)^T C_j^{-1} (v - m_j)$$

Segmental K-means

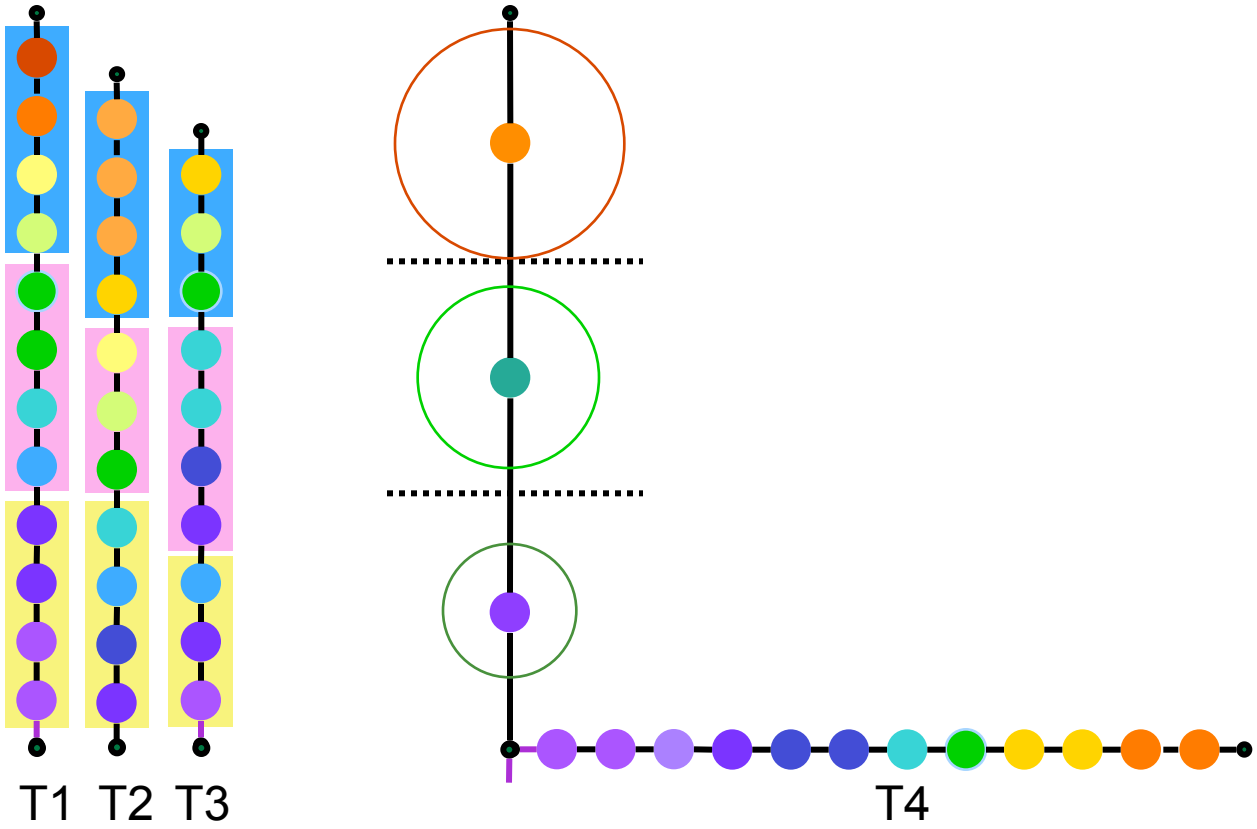
- Simple uniform segmentation of training instances is not the most effective method of grouping vectors in the training sequences
- A better segmentation strategy is to segment the training sequences such that the vectors within any segment are most alike
  - The total distance of vectors within each segment from the model vector for that segment is minimum
- This segmentation must be estimated
- The segmental K-means procedure is an iterative procedure to estimate the optimal segmentation

Alignment for training a model from multiple vector sequences



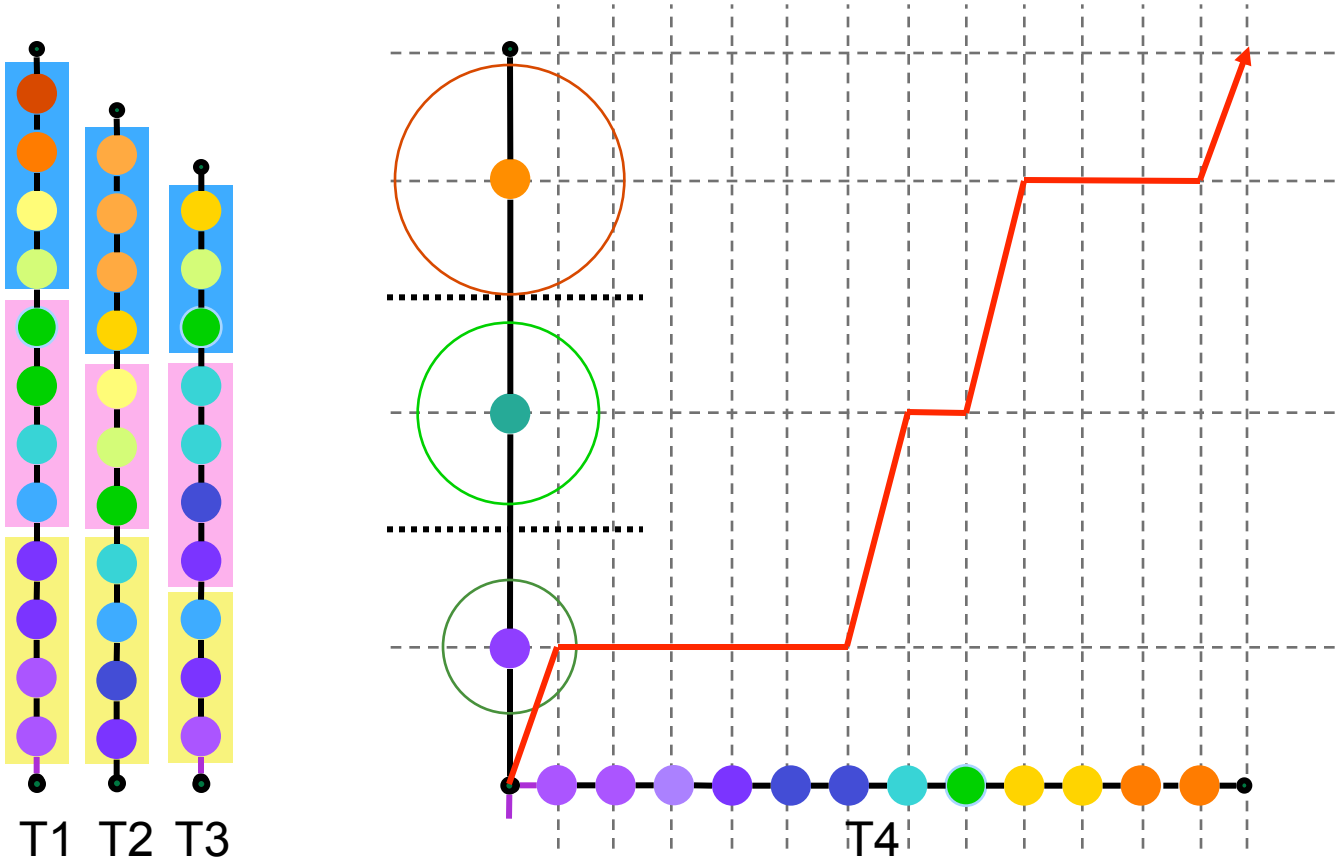
Initialize by uniform segmentation

Alignment for training a model from multiple vector sequences



Initialize by uniform segmentation

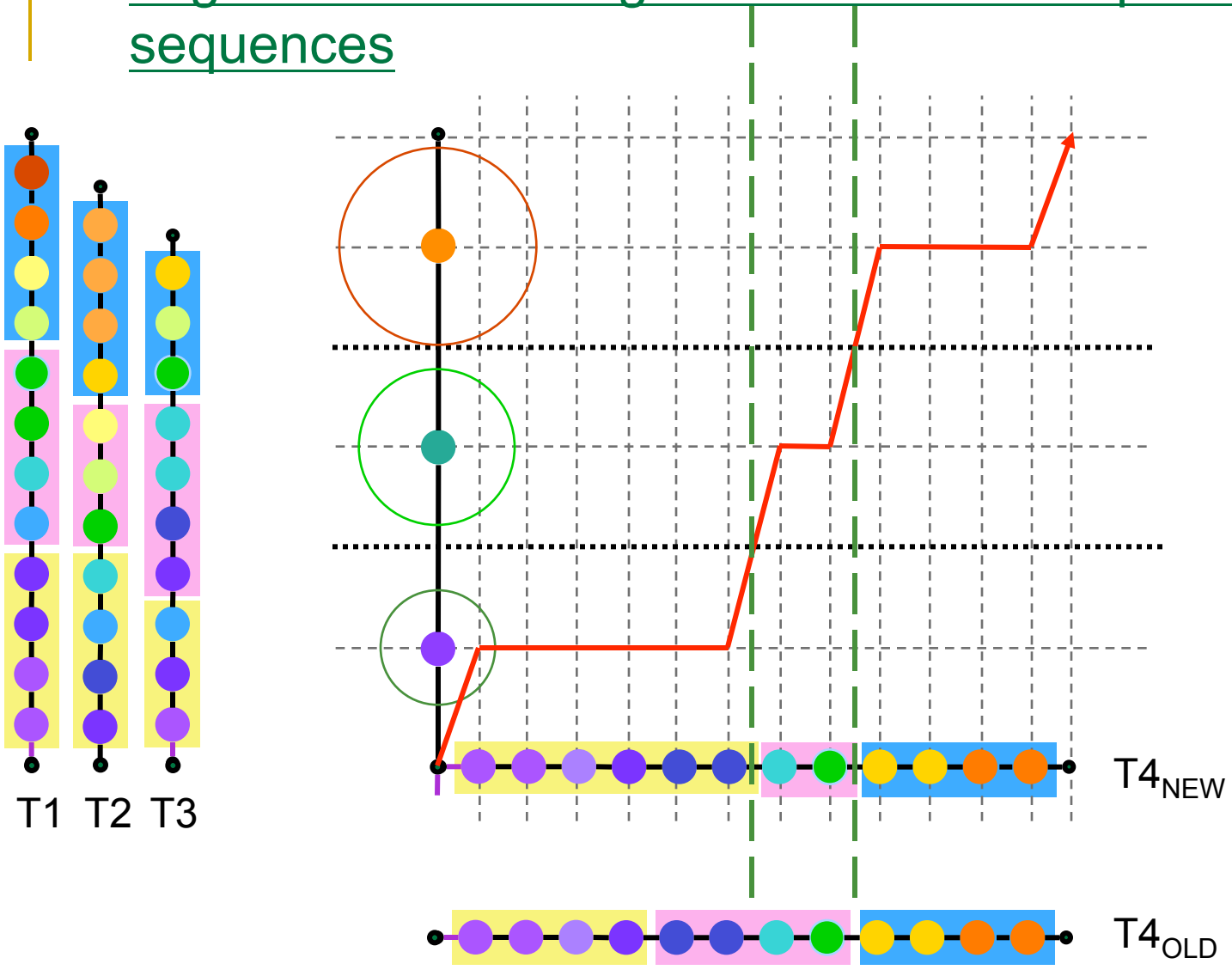
Alignment for training a model from multiple vector sequences



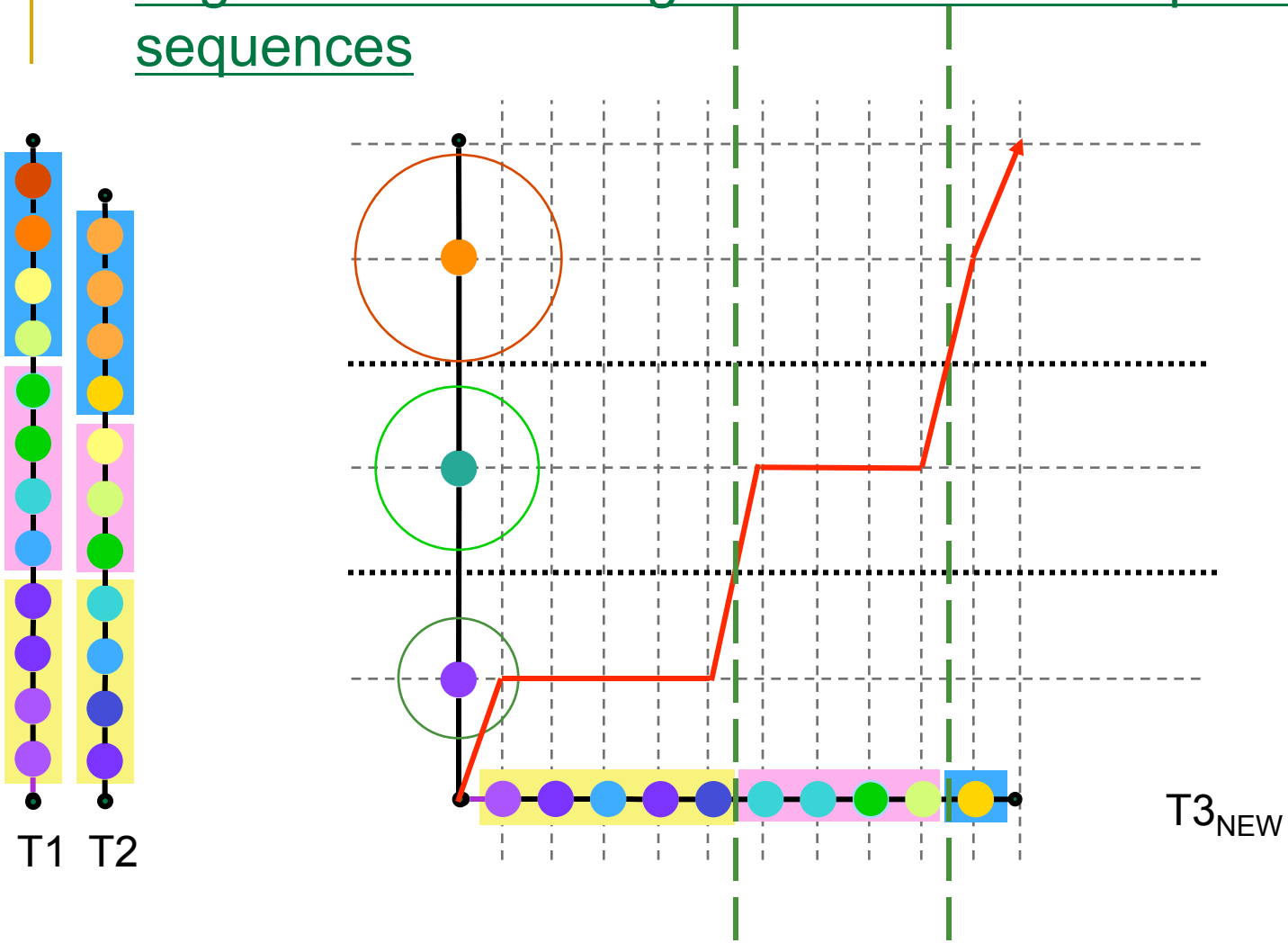
Initialize by uniform segmentation

Align each template to the averaged model to get new segmentations

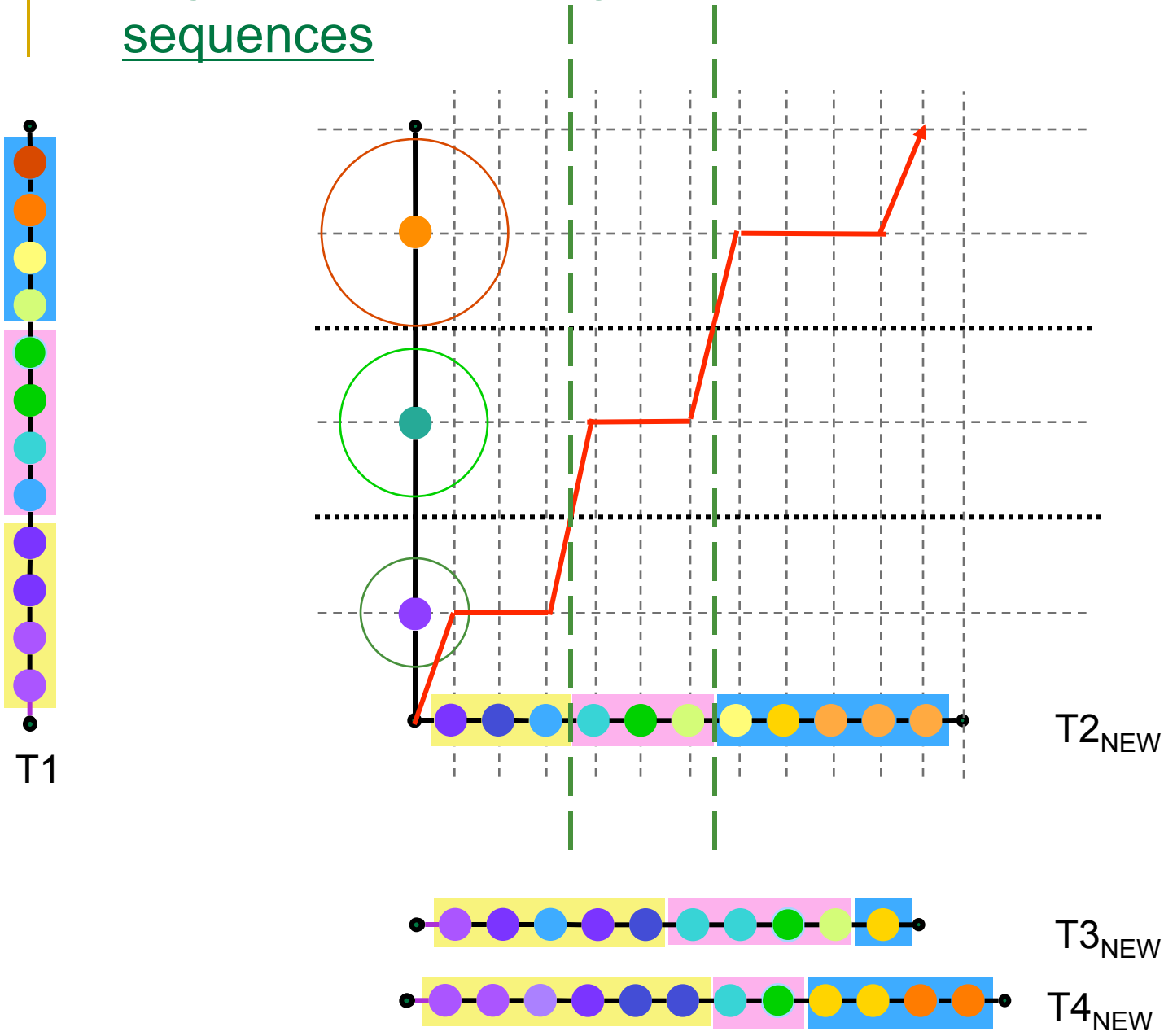
Alignment for training a model from multiple vector sequences



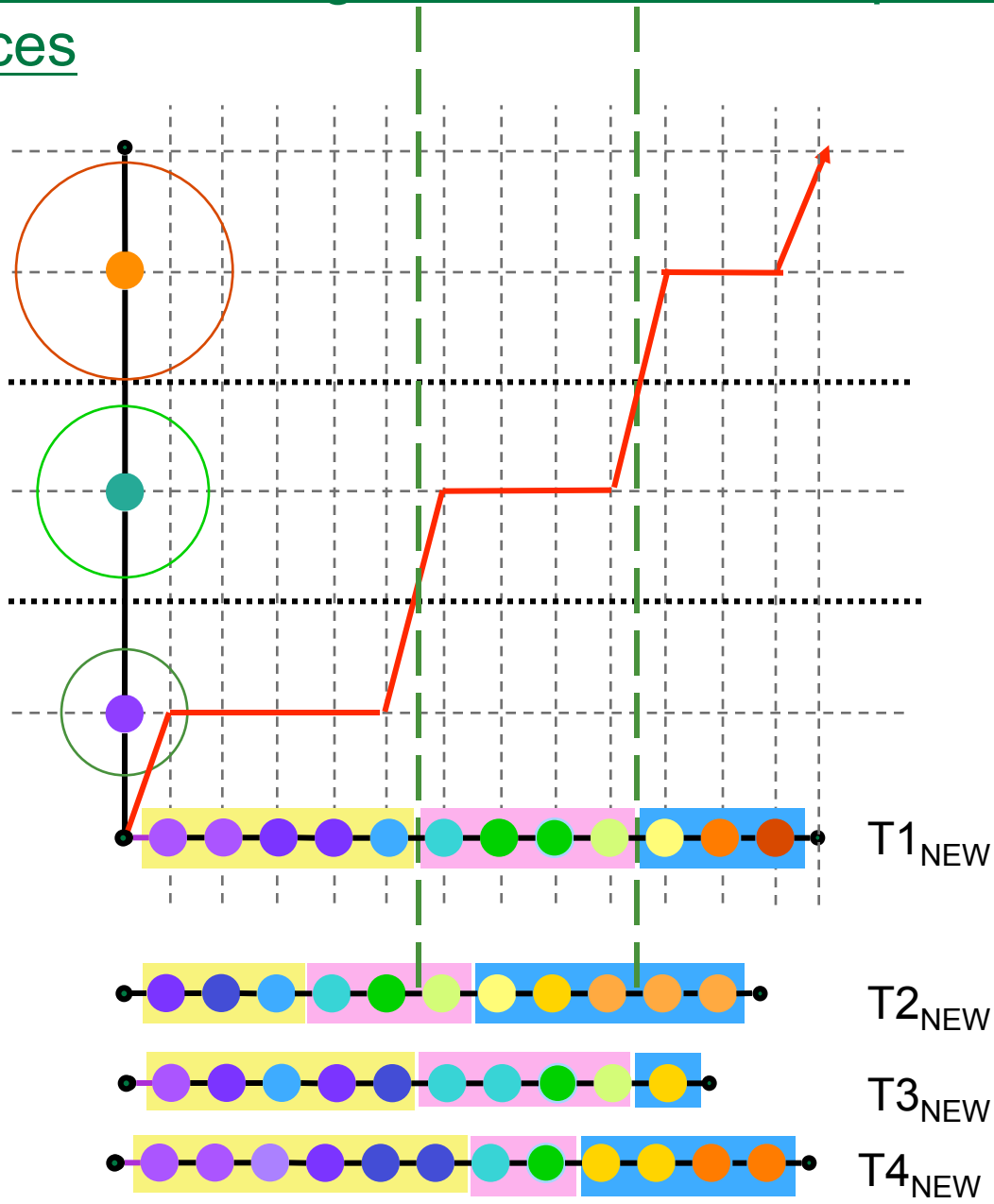
Alignment for training a model from multiple vector sequences



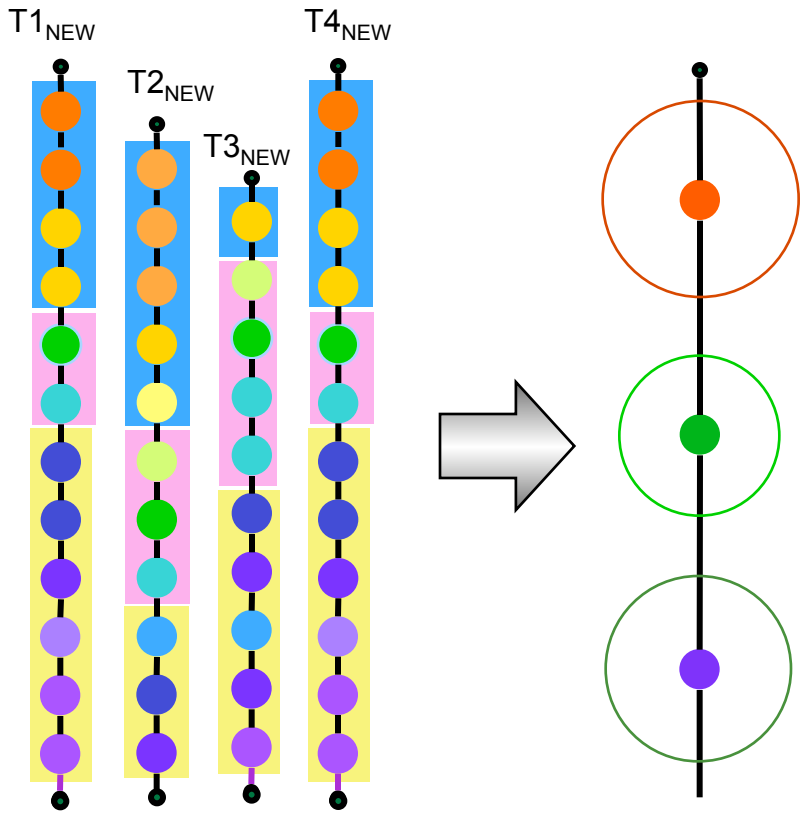
Alignment for training a model from multiple vector sequences



Alignment for training a model from multiple vector sequences

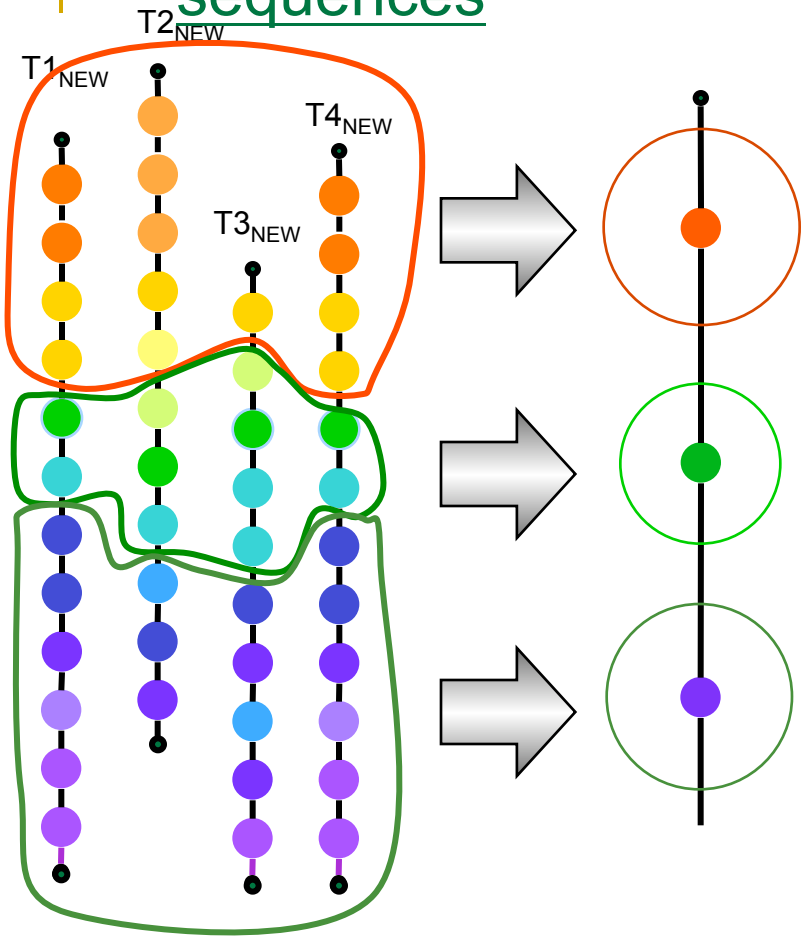


Alignment for training a model from multiple vector sequences

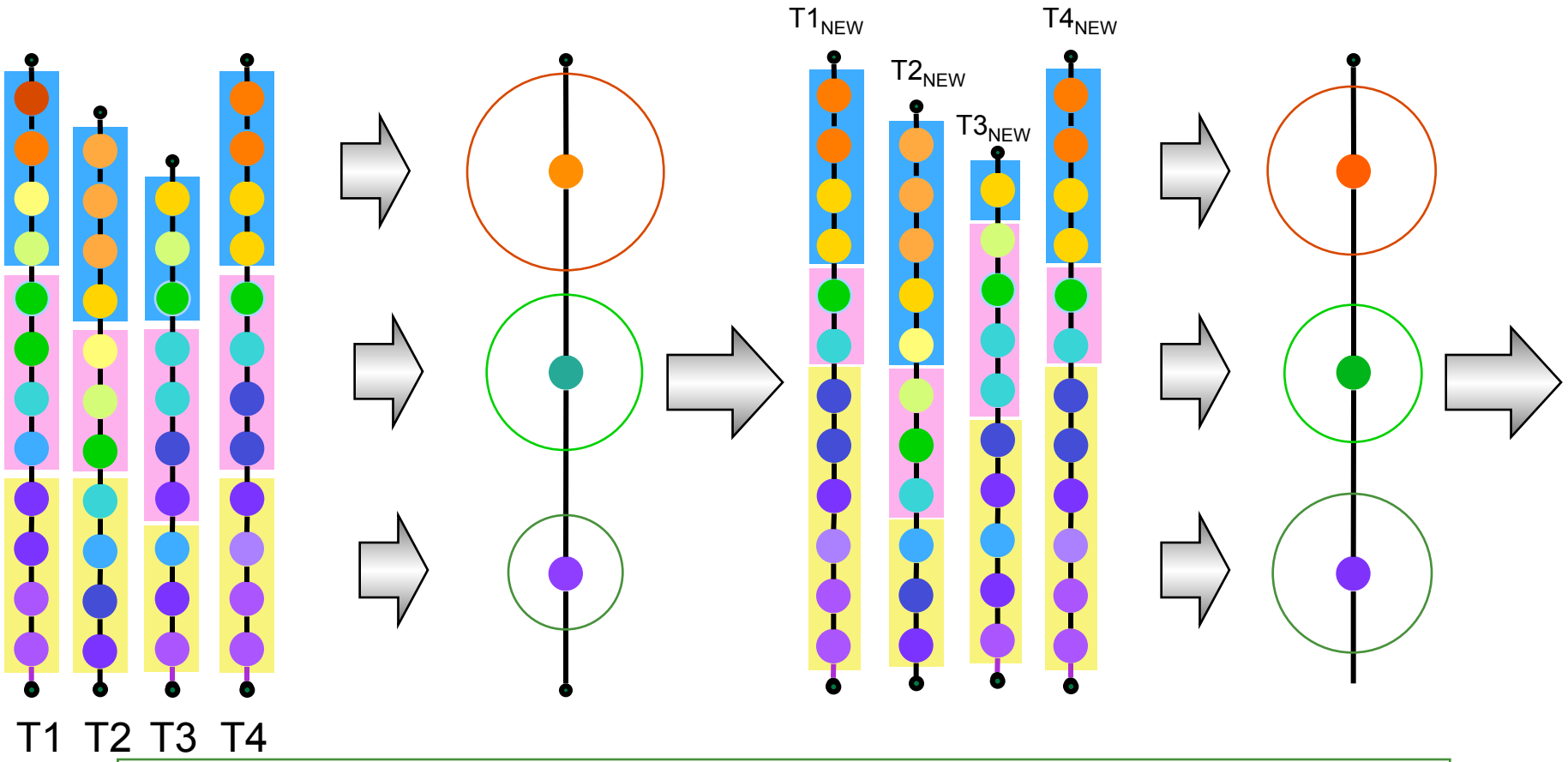


Initialize by uniform segmentation
Align each template to the averaged model to get new segmentations
Recompute the average model from new segmentations

Alignment for training a model from multiple vector sequences



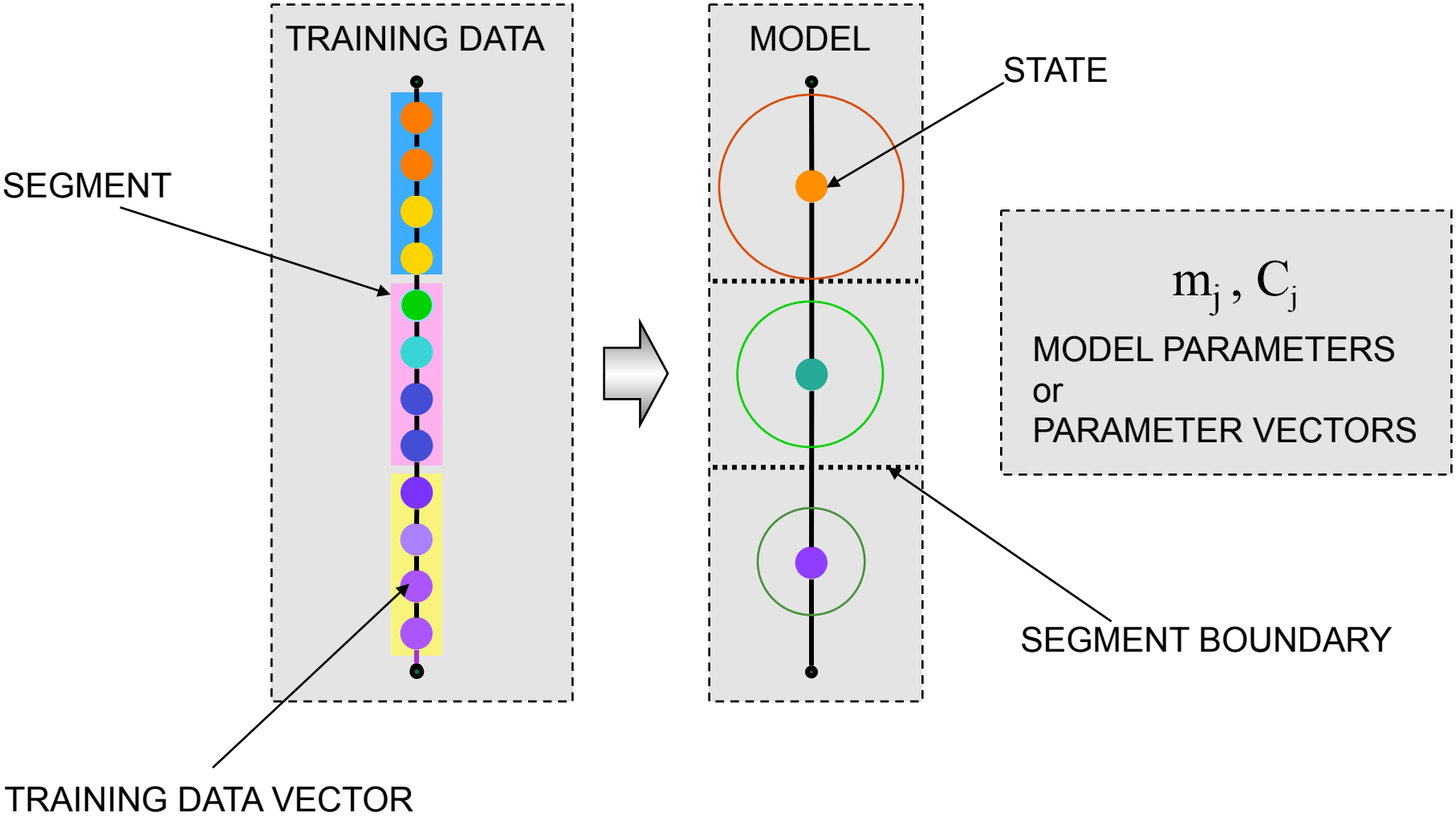
Alignment for training a model from multiple vector sequences



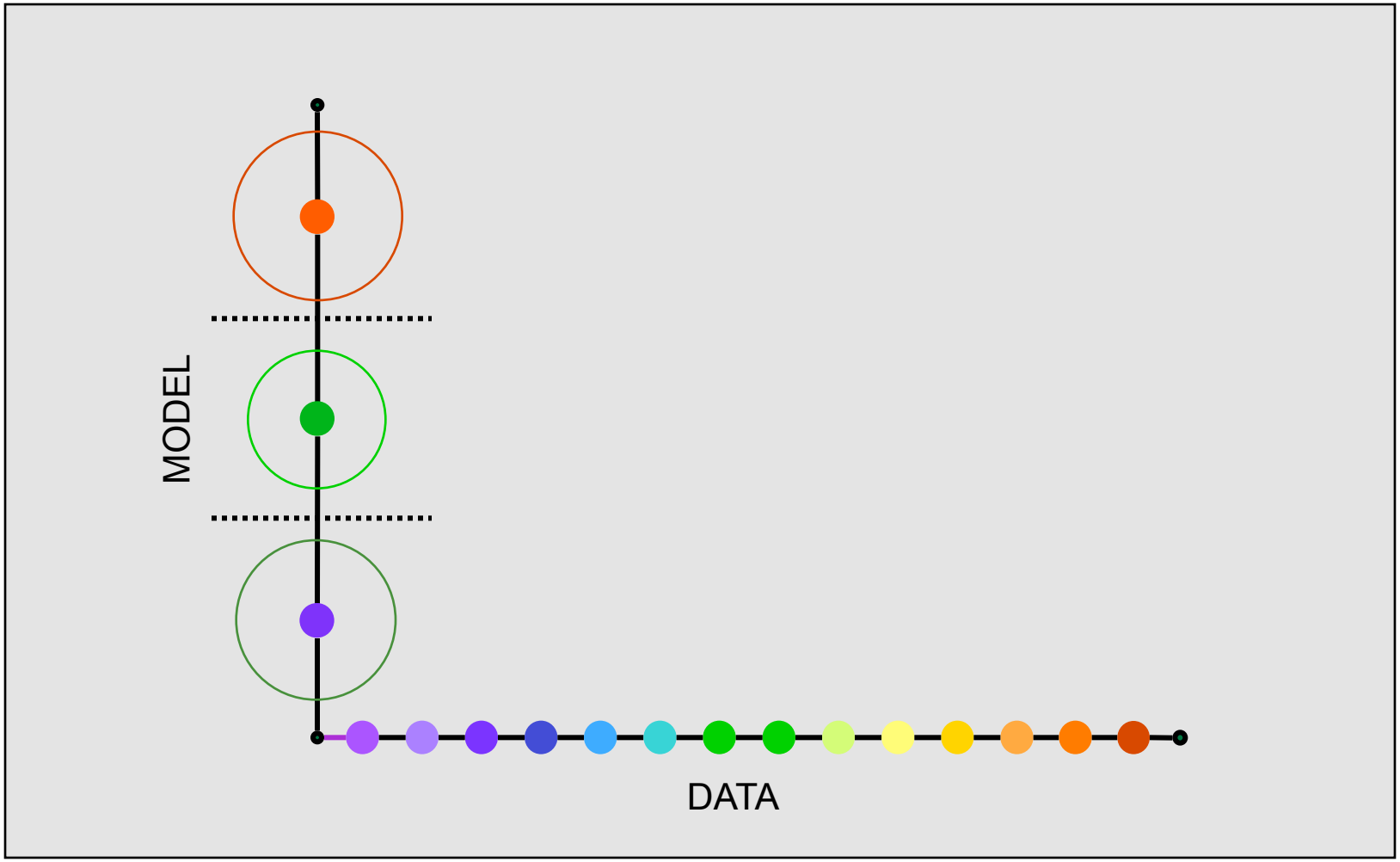
The procedure can be continued until convergence

Convergence is achieved when the total best-alignment error for all training sequences does not change significantly with further refinement of the model

Shifted terminology



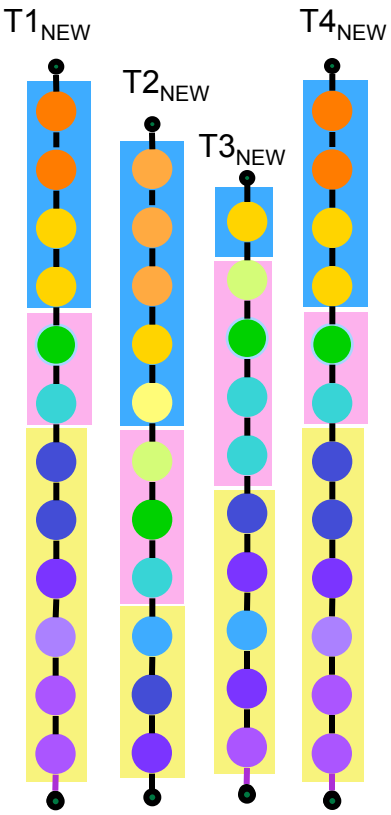
Transition structures in models



The converged models can be used to score / align data sequences

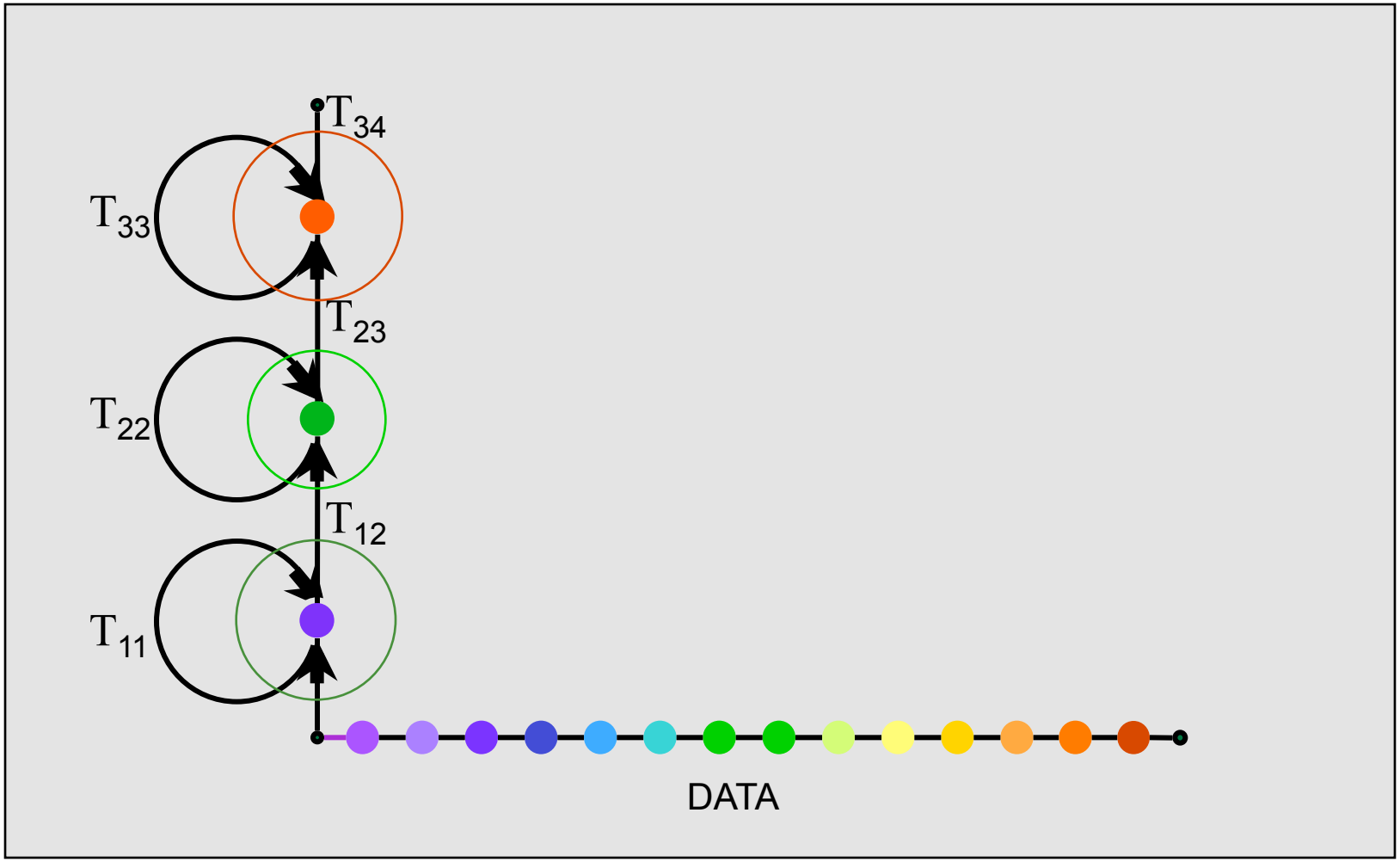
Model structure is incomplete.

DTW with multiple models



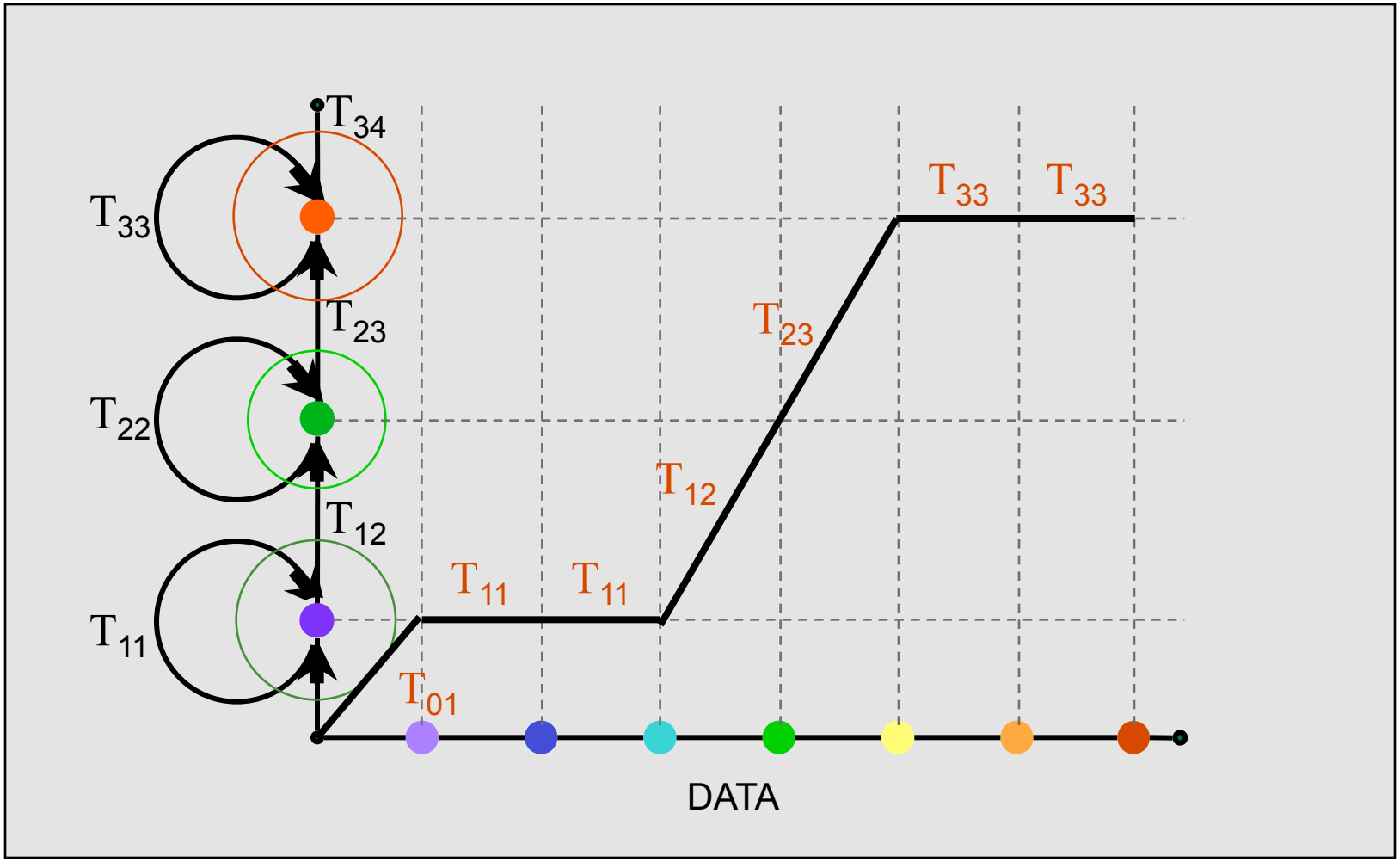
- Some segments are naturally longer than others
  - E.g., in the example the initial (yellow) segments are usually longer than the second (pink) segments
- This difference in segment lengths is different from the *variation* within a segment
  - Segments with small variance could still persist very long for a particular sound or word
- The DTW algorithm must account for these natural differences in typical segment length
- This can be done by having a state specific insertion penalty
  - States that have lower insertion penalties persist longer and result in longer segments

Transition structures in models



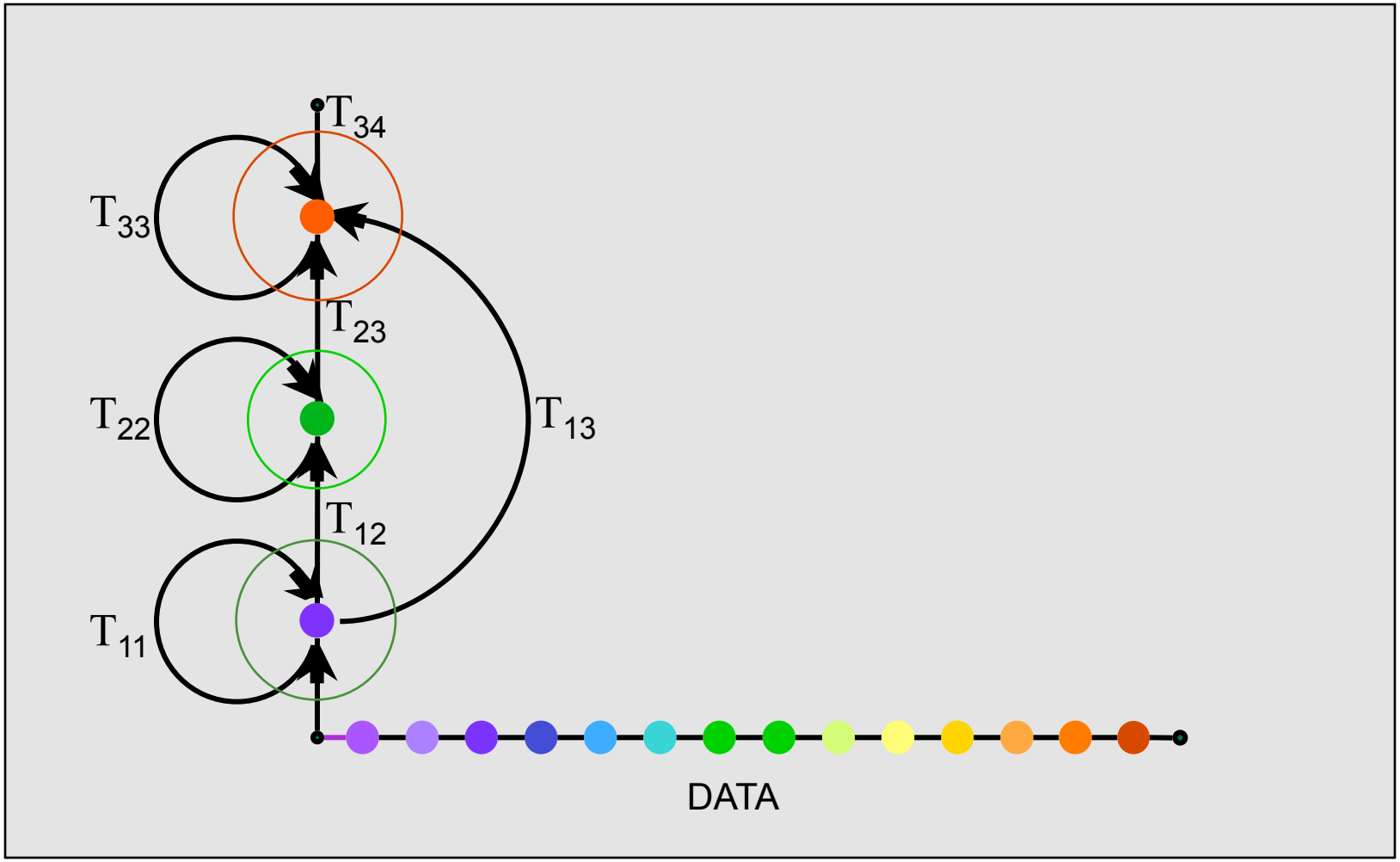
State specific insertion penalties are represented as self transition arcs for model vectors. Horizontal edges within the trellis will incur a penalty associated with the corresponding arc. Every transition within the model can have its own penalty.

Transition structures in models



State specific insertion penalties are represented as self transition arcs for model vectors. Horizontal edges within the trellis will incur a penalty associated with the corresponding arc. Every transition within the model can have its own penalty or score

Transition structures in models



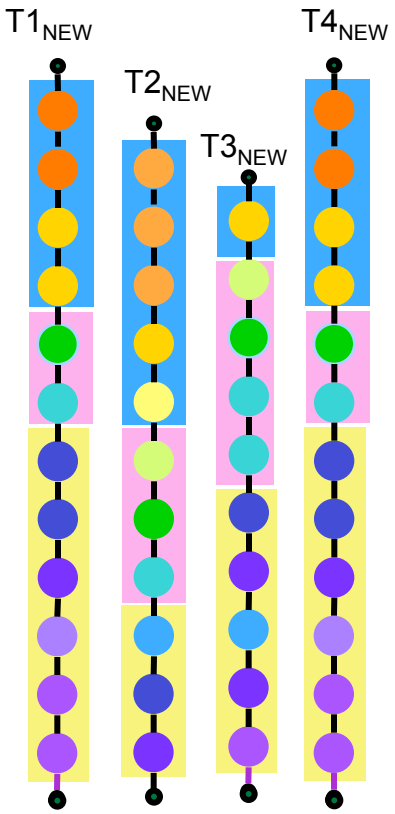
This structure also allows the inclusion of arcs that permit the central state to be skipped (deleted)

Other transitions such as returning to the first state from the last state can be permitted by inclusion of appropriate arcs

What should the transition scores be

- Transition behavior can be expressed with probabilities
  - For segments that are typically long, if a data vector is within that segment, the probability that the next vector will also be within it is high
- A good choice for transition scores are the negative logarithm of the probabilities of the appropriate transitions
  - T_{ij} is the negative of the log probability that if the current data vector belongs to the i^{th} state, the next data vector belongs to the j^{th} state
- More probable transitions are less penalized. Impossible transitions are infinitely penalized

Modified segmental K-means AKA Viterbi training

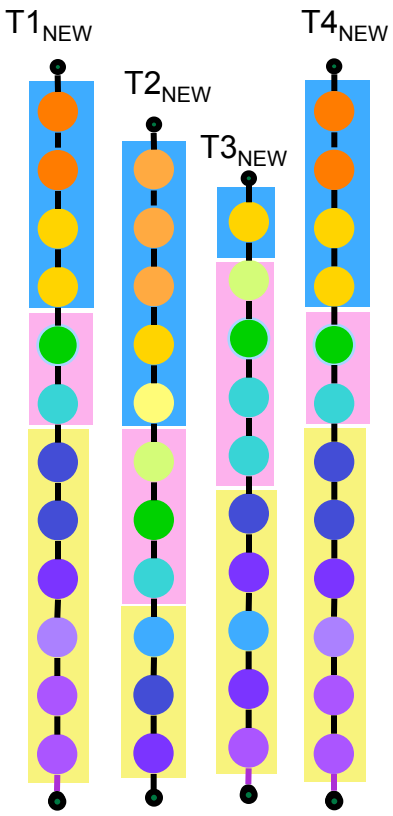


- Transition scores can be computed by a simple extension of the segmental K-means algorithm
- Probabilities can be counted by simple counting

$$P_{ij} = \frac{\sum_k N_{k,i,j}}{\sum_k N_{k,i}} \qquad T_{ij} = -\log(P_{ij})$$

- $N_{k,i}$ is the number of vectors in the i^{th} segment (state) of the k^{th} training sequence
- $N_{k,i,j}$ is the number of vectors in the i^{th} segment (state) of the k^{th} training sequence that were followed by vectors from the j^{th} segment (state)
  - E.g., No. of vectors in the 1st (yellow) state = 20
    No of vectors from the 1st state that were followed by vectors from the 1st state = 16
    $P_{11} = 16/20 = 0.8$; $T_{11} = -\log(0.8)$

Modified segmental K-means AKA Viterbi training



N = 4
N_{01} = 4
N_{02} = 0
N_{03} = 0

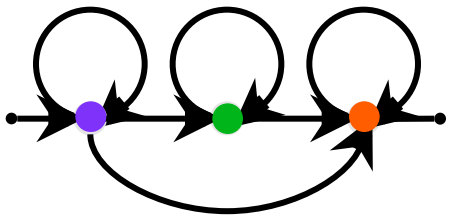
- A special score is the penalty associated with *starting* at a particular state
- In our examples we always begin at the first state
- Enforcing this is equivalent to setting $T_{01} = 0$, T_{0j} = infinity for j != 1
- It is sometimes useful to permit entry directly into later states
  - i.e. permit deletion of initial states
- The score for direct entry into any state can be computed as

$$P_j = \frac{N_{0j}}{N} \qquad T_{0j} = -\log(P_j)$$

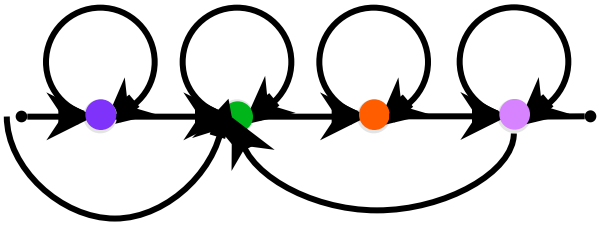
- N is the total number of training sequences
- N_{0j} is the number of training sequences for which the first data vector was in the j^{th} state

Modified segmental K-means AKA Viterbi training

- Some structural information must be prespecified
- The number of states must be prespecified
  - Manually
- Allowable start states and transitions must be presecified
  - E.g. we may specify beforehand that the first vector may be in states 1 or 2, but not 3
  - We may specify possible transitions between states



3 model vectors
Permitted initial states: 1
Permitted transitions: shown by arrows



4 model vectors
Permitted initial states: 1, 2
Permitted transitions: shown by arrows

Some example specifications

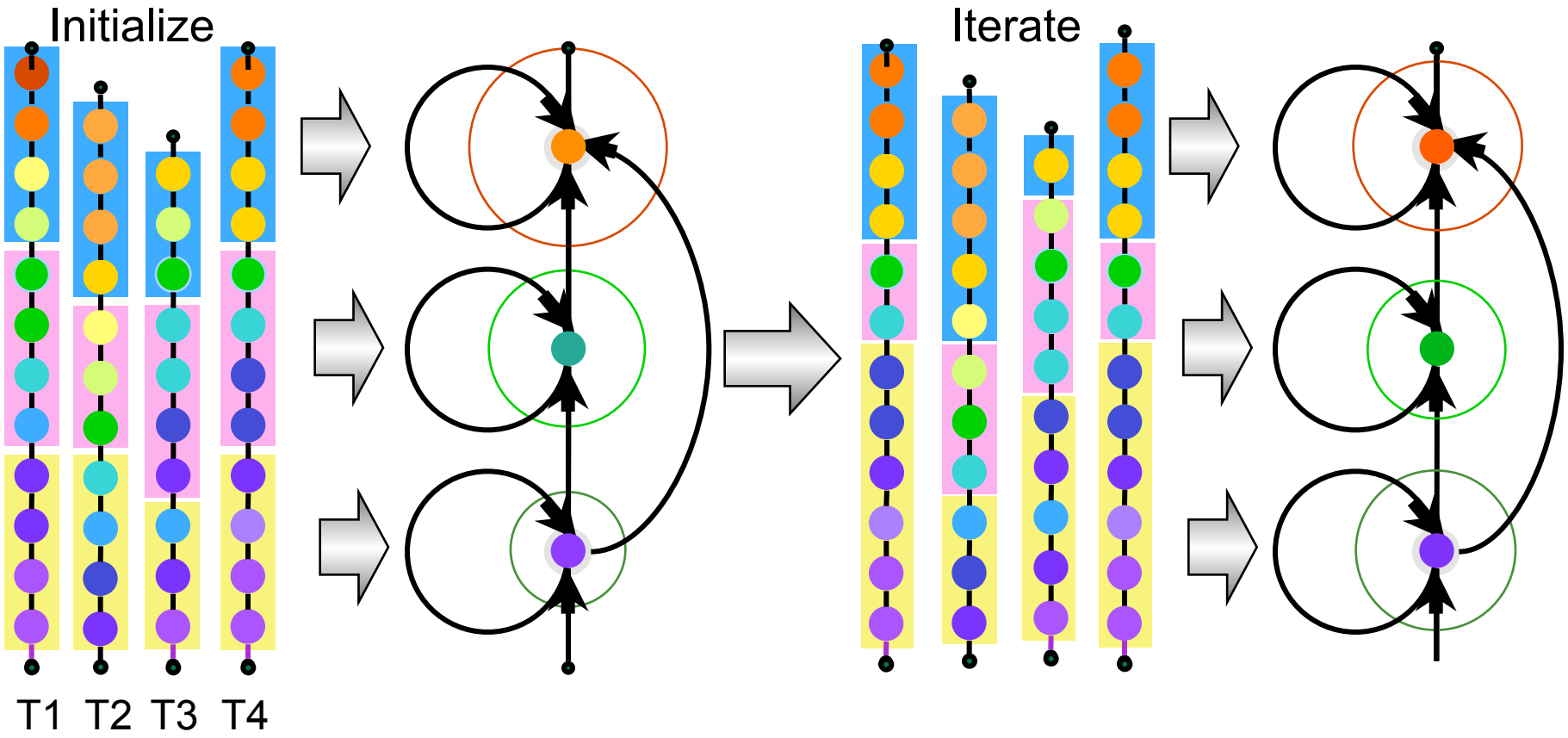
Modified segmental K-means AKA Viterbi training

- Initializing state parameters
  - Segment all training instances uniformly, learn means and variances
- Initializing T_{0j} scores
  - Count the number of permitted initial states
    - Let this number be M_0
  - Set all permitted initial states to be equiprobable: $P_j = 1/M_0$
  - $T_{0j} = -\log(P_j) = \log(M_0)$
- Initializing T_{ij} scores
  - For every state i, count the number of states that are permitted to follow
    - i.e. the number of arcs out of the state, in the specification
    - Let this number be Mi
  - Set all permitted transitions to be equiprobable: $P_{ij} = 1/M_i$
  - Initialize $T_{ij} = -\log(P_{ij}) = \log(M_i)$
- This is only one technique for initialization
  - Other methods possible, e.g. random initialization

Modified segmental K-means AKA Viterbi training

- The entire segmental K-means algorithm:
  - Initialize all parameters
    - State means and covariances
    - Transition scores
    - Entry transition scores
  - Segment all training sequences
  - Reestimate parameters from segmented training sequences
  - If not converged, return to 2

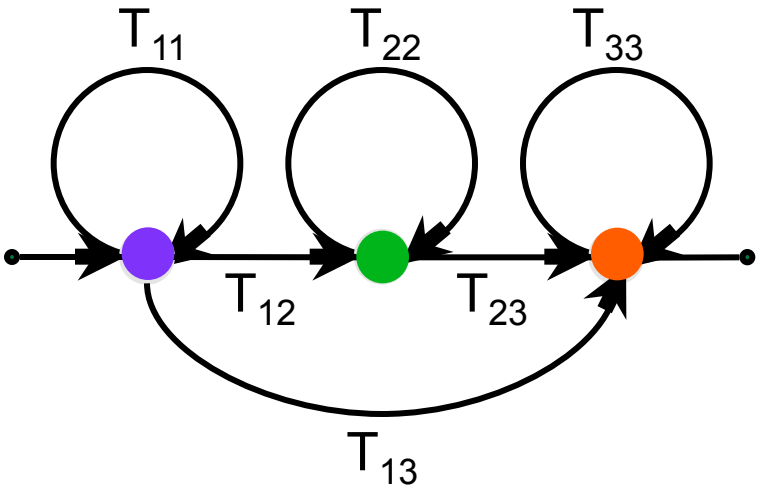
Alignment for training a model from multiple vector sequences



The procedure can be continued until convergence

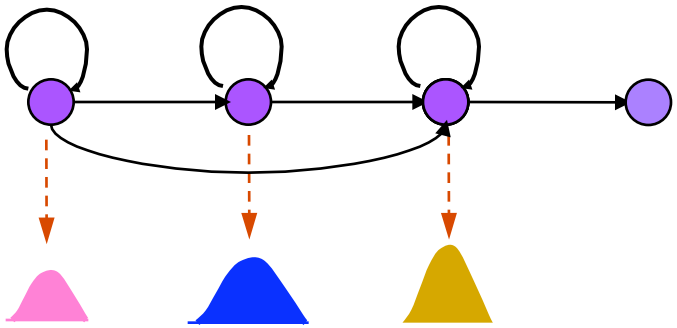
Convergence is achieved when the total best-alignment error for all training sequences coverges

DTW and Hidden Markov Models (HMMs)

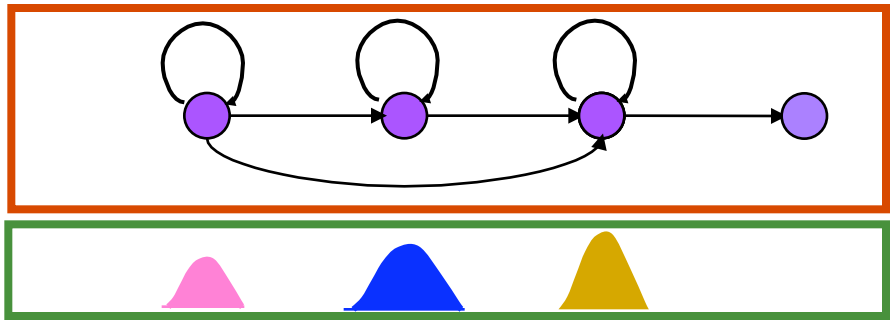


- This structure is a generic representation of a statistical model for processes that generate time series
- The “segments” in the time series are referred to as states
  - The process passes through these states to generate time series
- The entire structure may be viewed as *one* generalization of the DTW models we have discussed thus far
- Strict left-to-right Bakis topology

Hidden Markov Models



- A Hidden Markov Model consists of two components
  - A state/transition backbone that specifies how many states there are, and how they can follow one another
  - A set of probability distributions, one for each state, which specifies the distribution of all vectors in that state



- This can be factored into two separate probabilistic entities
  - A probabilistic Markov chain with states and transitions
  - A set of data probability distributions, associated with the states

HMMS and DTW

- HMMs are similar to DTW templates
  - DTW: Minimize negative log probability (cost)
  - HMM: Maximize probability
- In the models considered so far, the state output distribution have been assumed to be Gaussian
- In reality, the distribution of vectors within any state need not be Gaussian
  - In the most general case it can be arbitrarily complex
  - The Gaussian is only a coarse representation of this distribution
  - Typically they are Gaussian Mixtures
- Training algorithm: Baum Welch may replace segmental K-means
  - Segmental K-means is also quite effective

Gaussian Mixtures

- A Gaussian Mixture is literally a mixture of Gaussians. It is a weighted combination of several Gaussian distributions

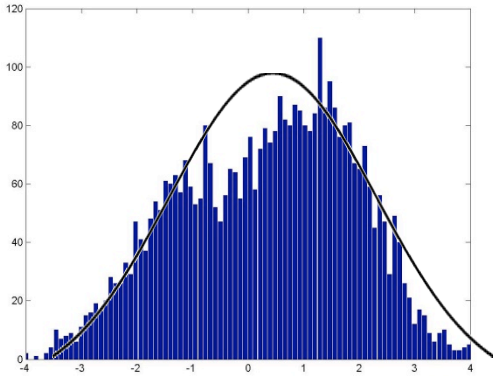
$$P(v) = \sum_{i=0}^{K-1} w_i Gaussian(v; m_i, C_i)$$

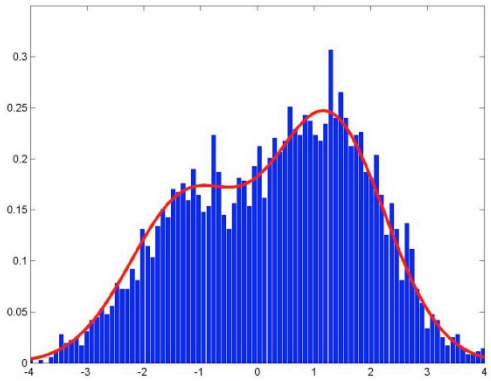
  - v is any data vector. P(v) is the probability given to that vector by the Gaussian mixture
  - K is the number of Gaussians being mixed
  - w_i is the mixture weight of the i^{th} Gaussian. m_i is its mean and C_i is its covariance

- Trained using all vectors in a segment
  - Instead of computing a single mean and covariance only, computes means and covariances of all Gaussians in the mixture

Gaussian Mixtures

- A Gaussian mixture can represent data distributions far better than a simple Gaussian
- The two panels show the histogram of an unknown random variable
- The first panel shows how it is modeled by a simple Gaussian
- The second panel models the histogram by a mixture of two Gaussians
- Caveat: It is hard to know the optimal number of Gaussians in a mixture distribution for any random variable





HMMS

- The parameters of an HMM with Gaussian mixture state distributions are:
  - π the set of initial state probabilities for all states
  - ***T*** the matrix of transition probabilities
  - A Gaussian mixture distribution for every state in the HMM. The Gaussian mixture for the i^{th} state is characterized by
    - K_i, the number of Gaussians in the mixture for the i^{th} state
    - The set of mixture weights $w_{i,j}$ $0<j<K_i$
    - The set of Gaussian means $m_{i,j}$ $0<j<K_i$
    - The set of Covariance matrices $C_{i,j}$ $0 < j <K_i$

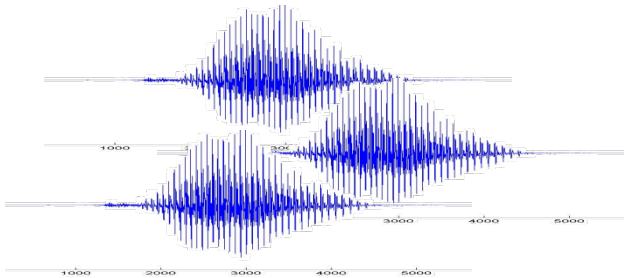
Segmenting and scoring data sequences with HMMs with Gaussian mixture state distributions

- The procedure is identical to what is used when state distributions are Gaussians with one minor modification:

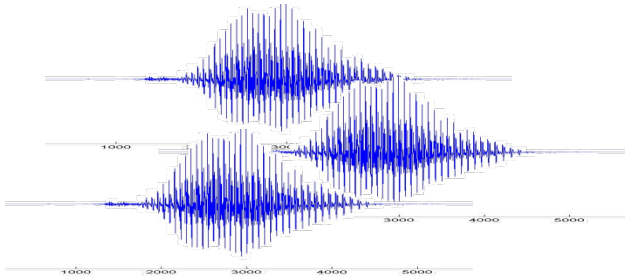
- The distance of any vector from a state is now the negative log of the probability given to the vector by the state distribution

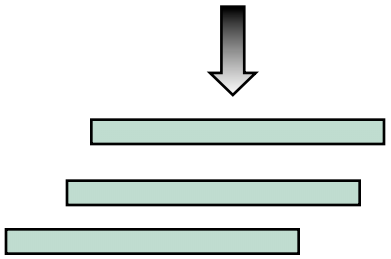
- The “penalty” applied to any transition is the negative log of the corresponding transition probability

Training word models

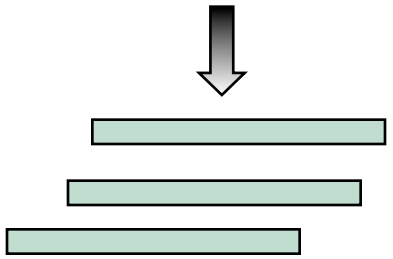


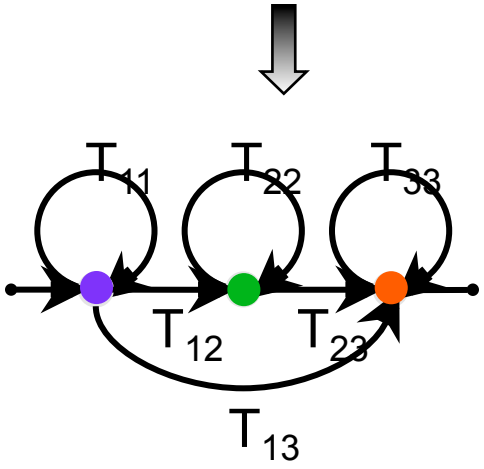
Record instances





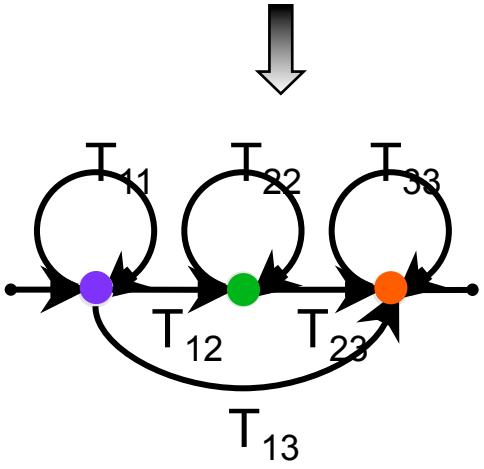
Compute features





Define model structure

- Specify number of states
- Specify transition structure
- Specify no. of Gaussians in the distribution of any state

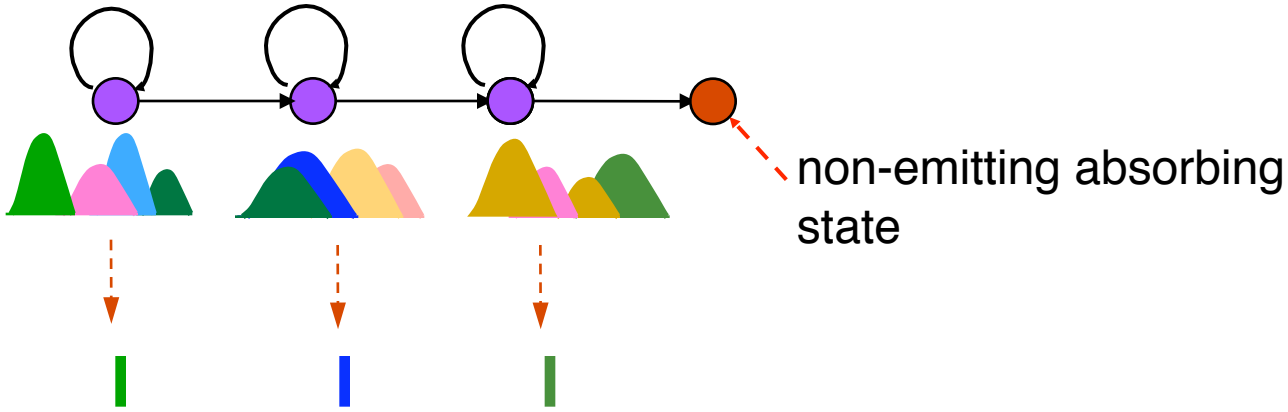


Train

- HMMs using segmental K-means.
- Mixture Gaussians for each state using K-means or EM

A Non-Emitting State

- A special kind of state: An NON-EMITTING state. No observations are generated from this state
- Usually used to model the termination of a unit



Statistical pattern classification

- Given data *X*, find which of a number of classes C_1, C_2,…C_N it belongs to, based on known distributions of data from C_1, C_2, etc.
- Bayesian Classification:

$$\text{Class} = C_i : i = \text{argmin}_j \; -\log(P(C_j)) - \log(P(X|C_j))$$

a priori probability of C_j

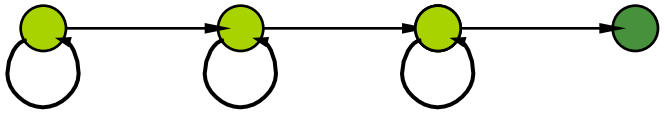
Probability of X as given by the probability distribution of C_j

- The *a priori* probability accounts for the relative proportions of the classes
  - If you never saw any data, you would guess the class based on these probabilities alone
- $P(X|C_j)$ accounts for evidence obtained from observed data X
  - -Log(P(X|C)) is approximated by the DTW score of the model

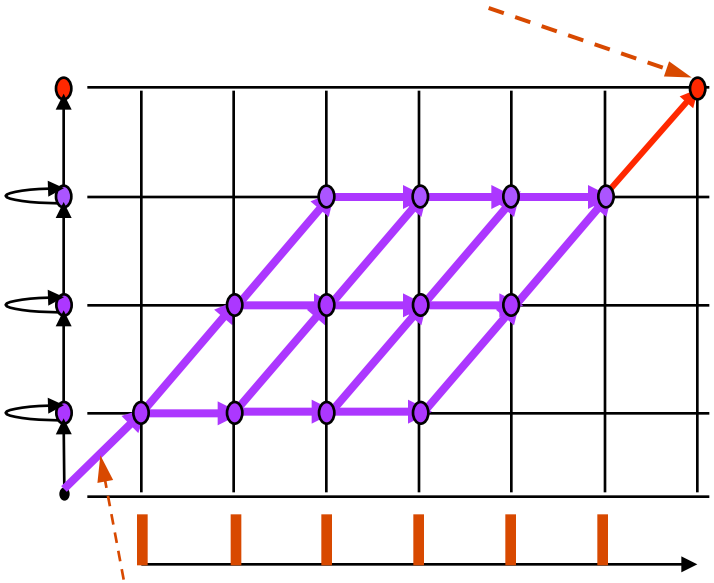
Classifying between two words: *Odd* and *Even*

HMM for *Odd*

HMM for *Even*

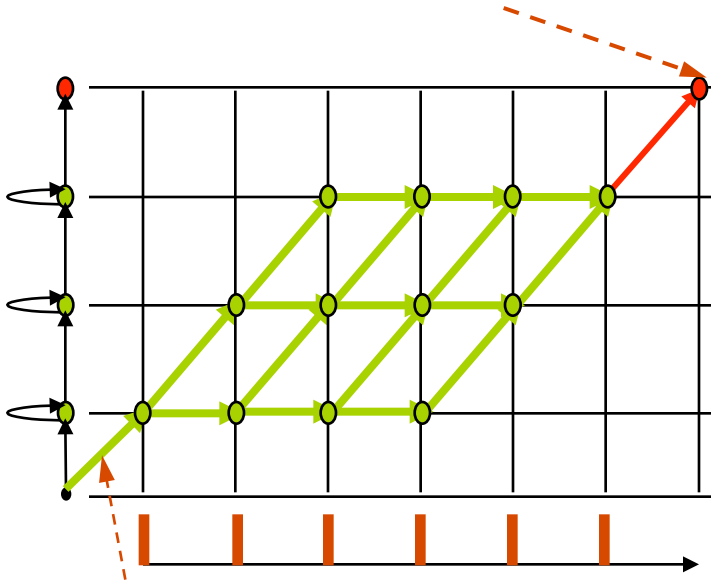


Log(P(*Odd*))+ log(P(*X*|*Odd*))



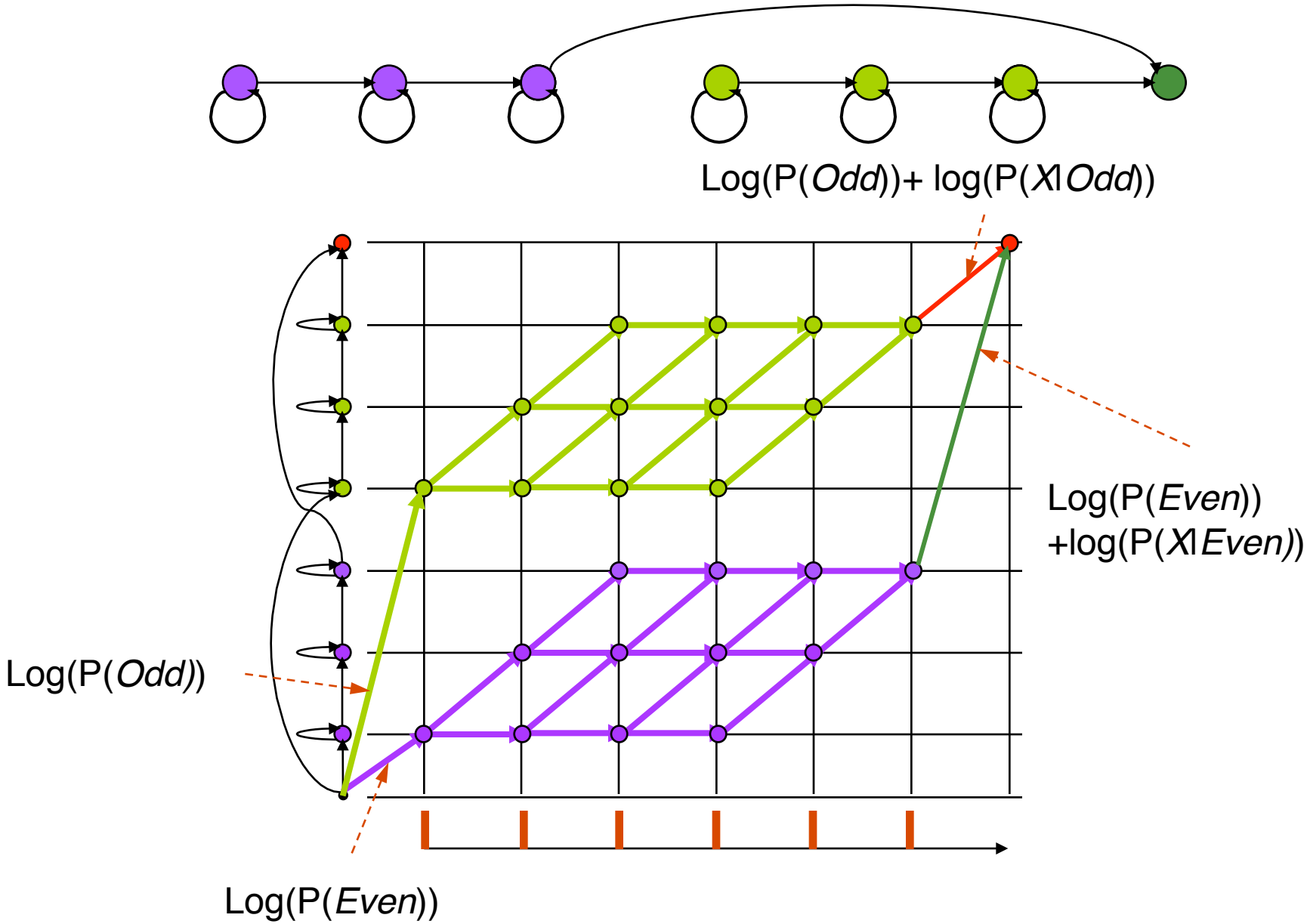
Log(P(*Odd*))

Log(P(*Even*))+log(P(*X*|*Even*))



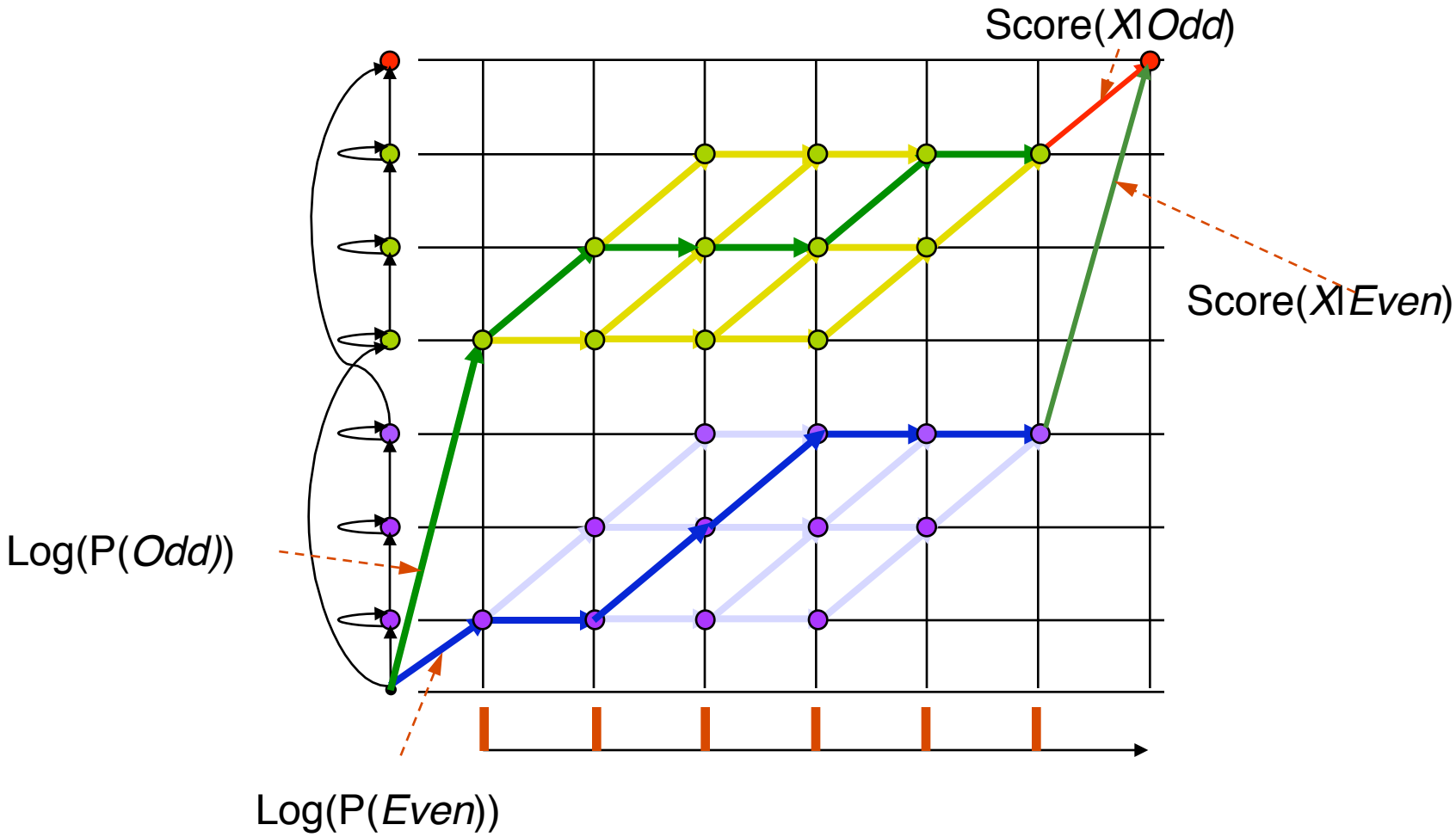
Log(P(*Even*))

Classifying between two words: *Odd* and *Even*



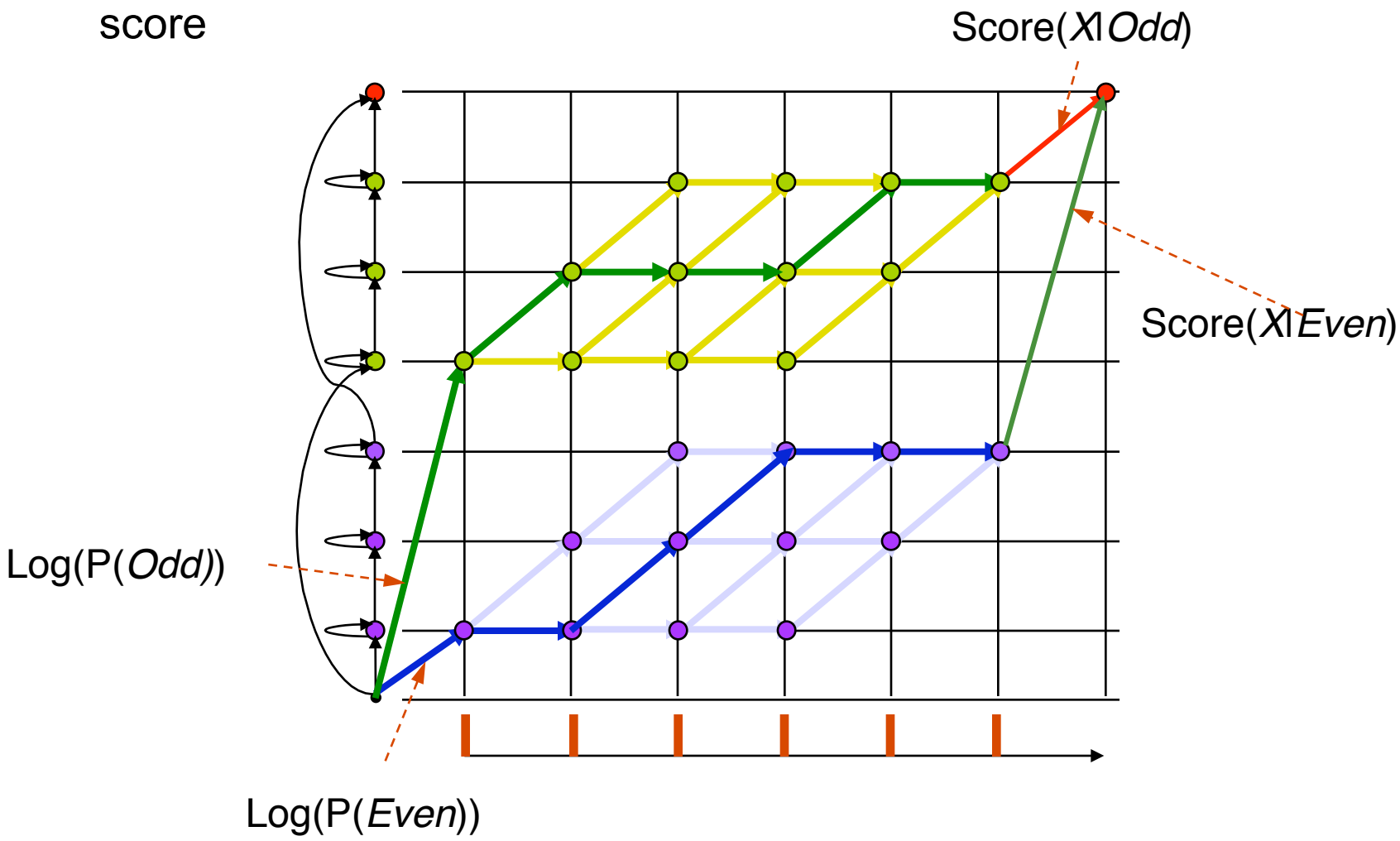
Decoding to classify between *Odd* and *Even*

- Compute the score of the best path



Decoding to classify between *Odd* and *Even*

- Compare scores (best state sequence probabilities) of all competing words
- Select the word sequence corresponding to the path with the best score



Statistical classification of word sequences

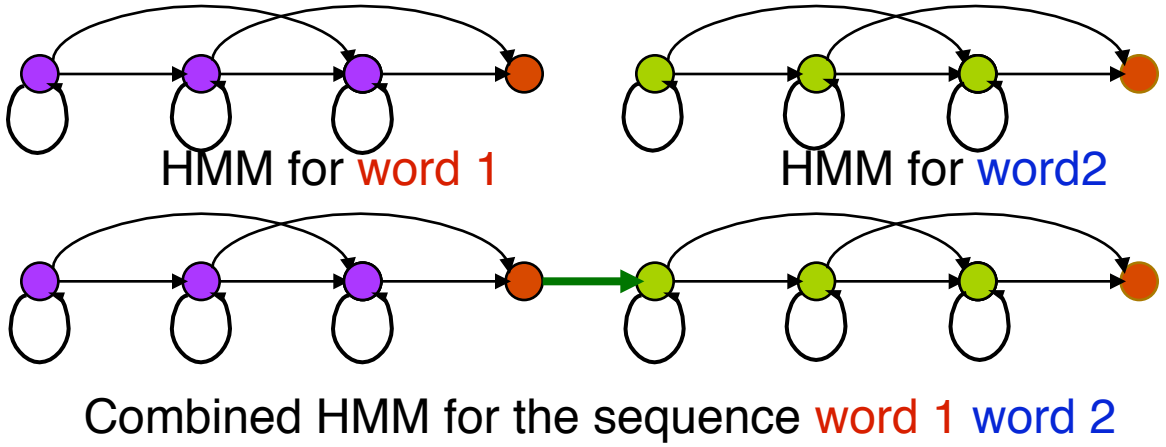
$$word_1, word_2, ..., word_N =$$
$$\arg\max_{wd_1, wd_2, ..., wd_N} \{P(X \mid wd_1, wd_2, ..., wd_N) P(wd_1, wd_2, ..., wd_N)\}$$

- P(wd_1,wd_2,wd_3..) is *a priori* probability of word sequence wd_1,wd_2,wd_3..
  - Obtained from a model of the language

- P(X| wd_1,wd_2,wd_3..) is the probability of X computed on the probability distribution function of the word sequence wd_1,wd_2,wd_3..
  - HMMs now represent probability distributions of word sequences

Decoding continuous speech

First step: construct an HMM for each possible word sequence

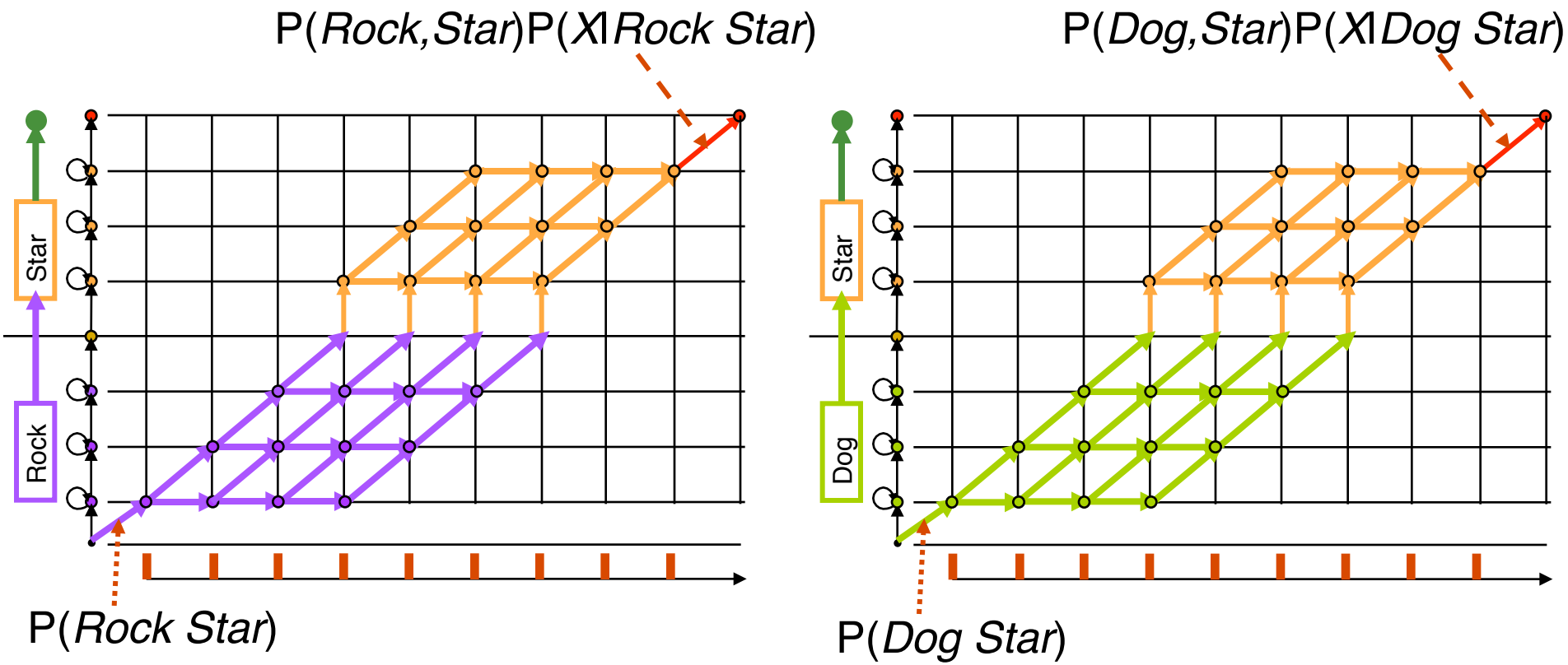


Second step: find the probability of the given utterance on the HMM for each possible word sequence

- P(X| wd_1,wd_2,wd_3..) is the probability of X computed on the probability distribution function of the word sequence wd_1,wd_2,wd_3..
  - HMMs now represent probability distributions of word sequences

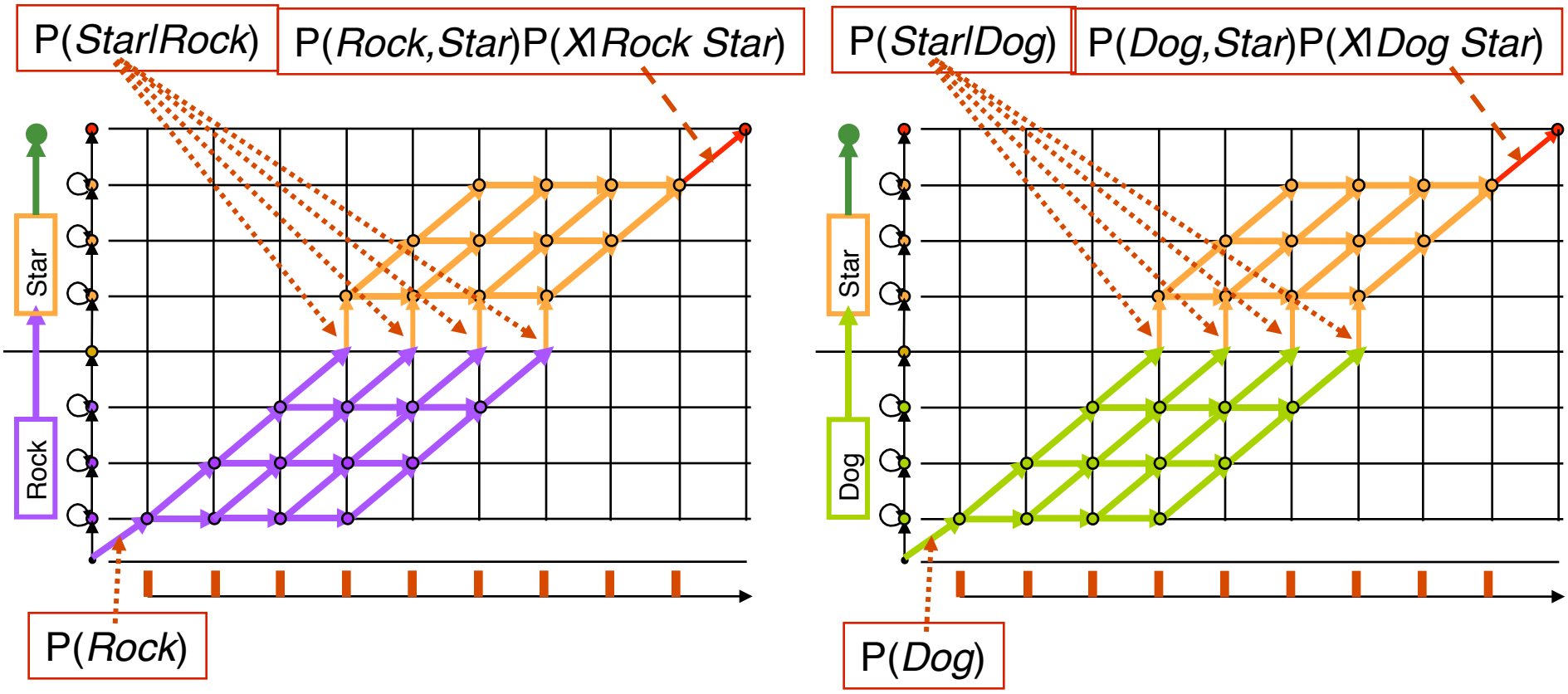
Bayesian Classification between word sequences

- Classifying an utterance as either "Rock Star" or "Dog Star"
  - Must compare P(Rock,Star)P(X|Rock Star) with P(Dog,Star)P(X|Dog Star)

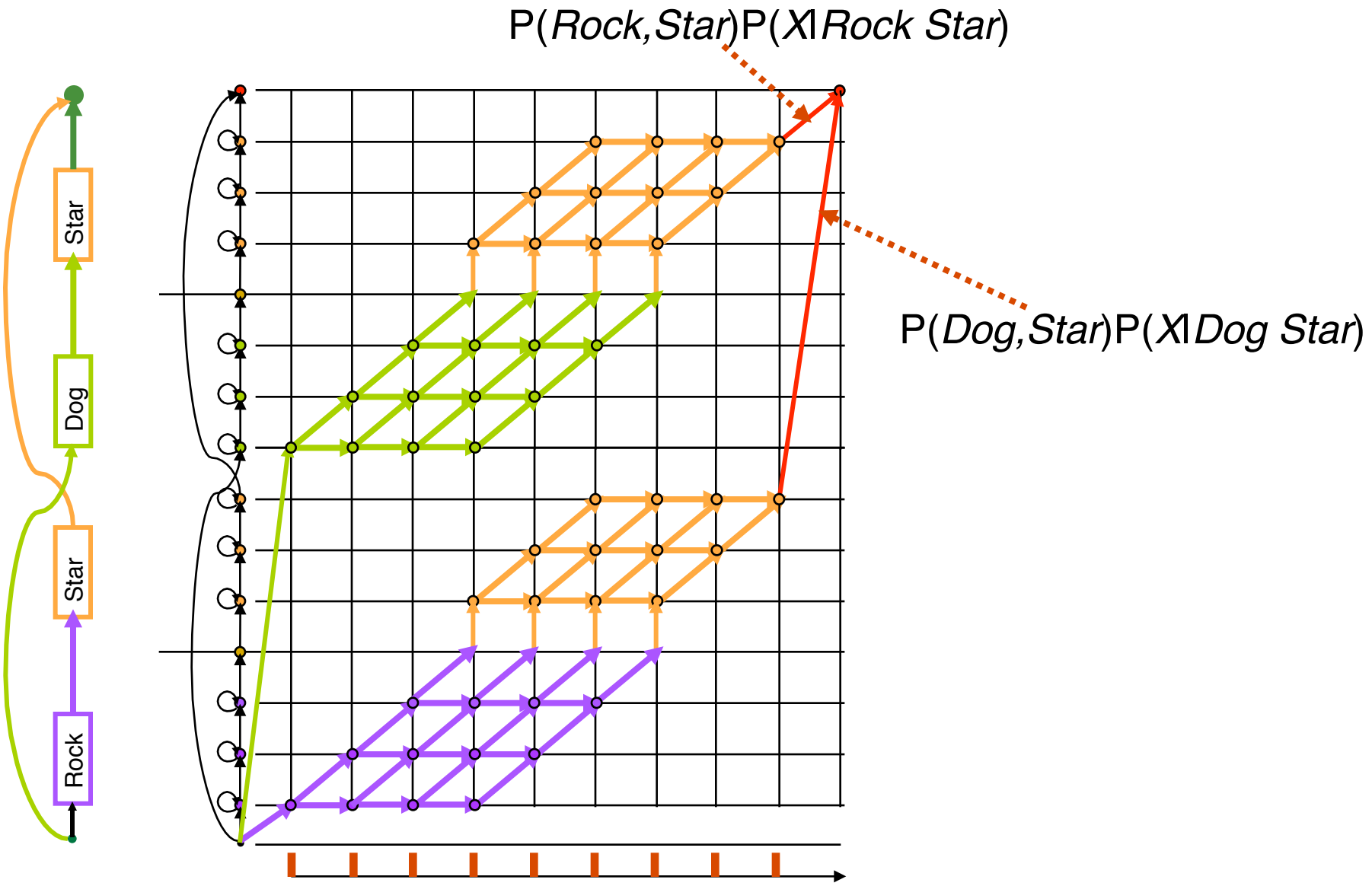


Bayesian Classification between word sequences

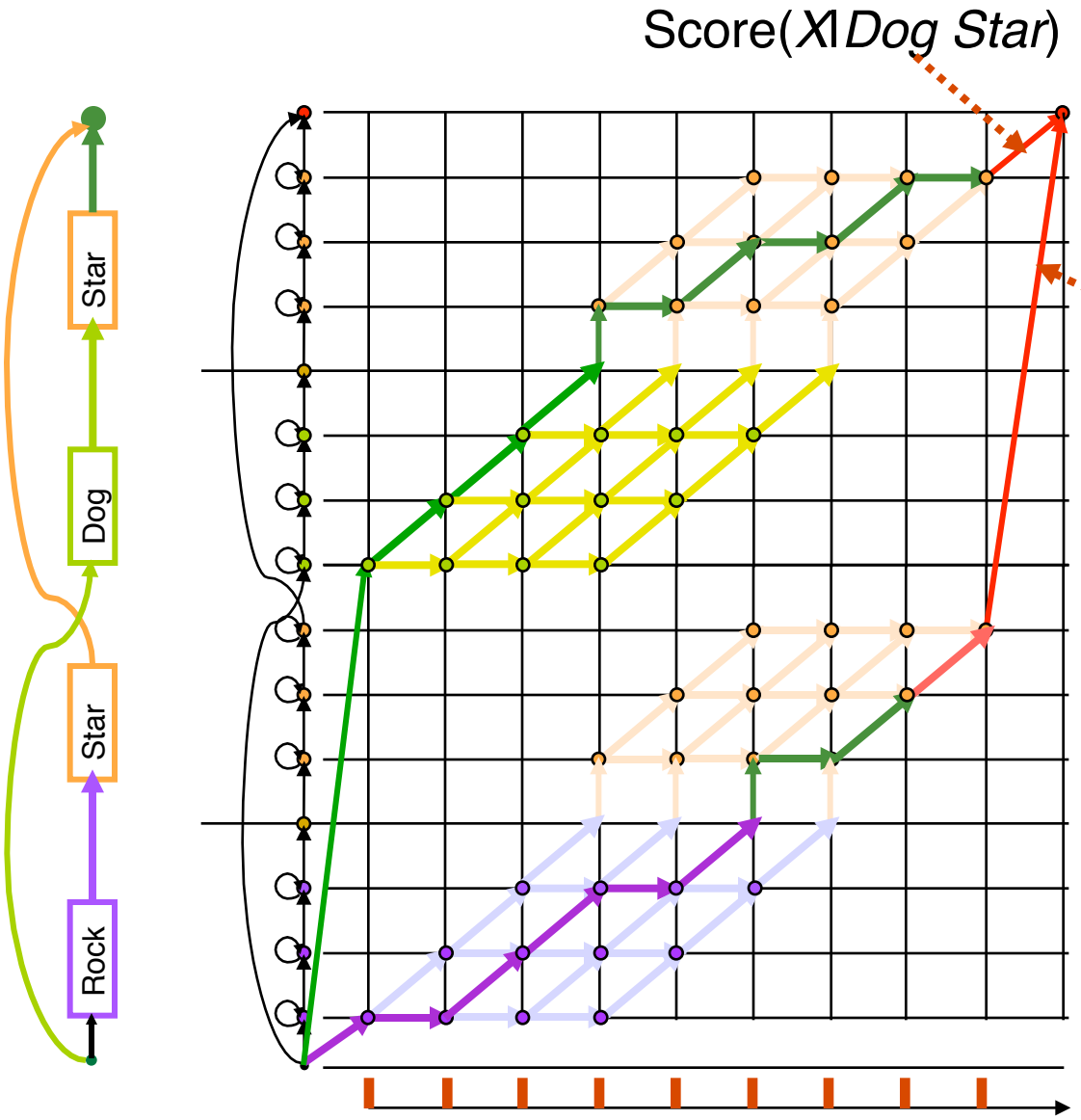
- Classifying an utterance as either "Rock Star" or "Dog Star"
  - Must compare P(Rock,Star)P(X|Rock Star) with P(Dog,Star)P(X|Dog Star)



Bayesian Classification between word sequences

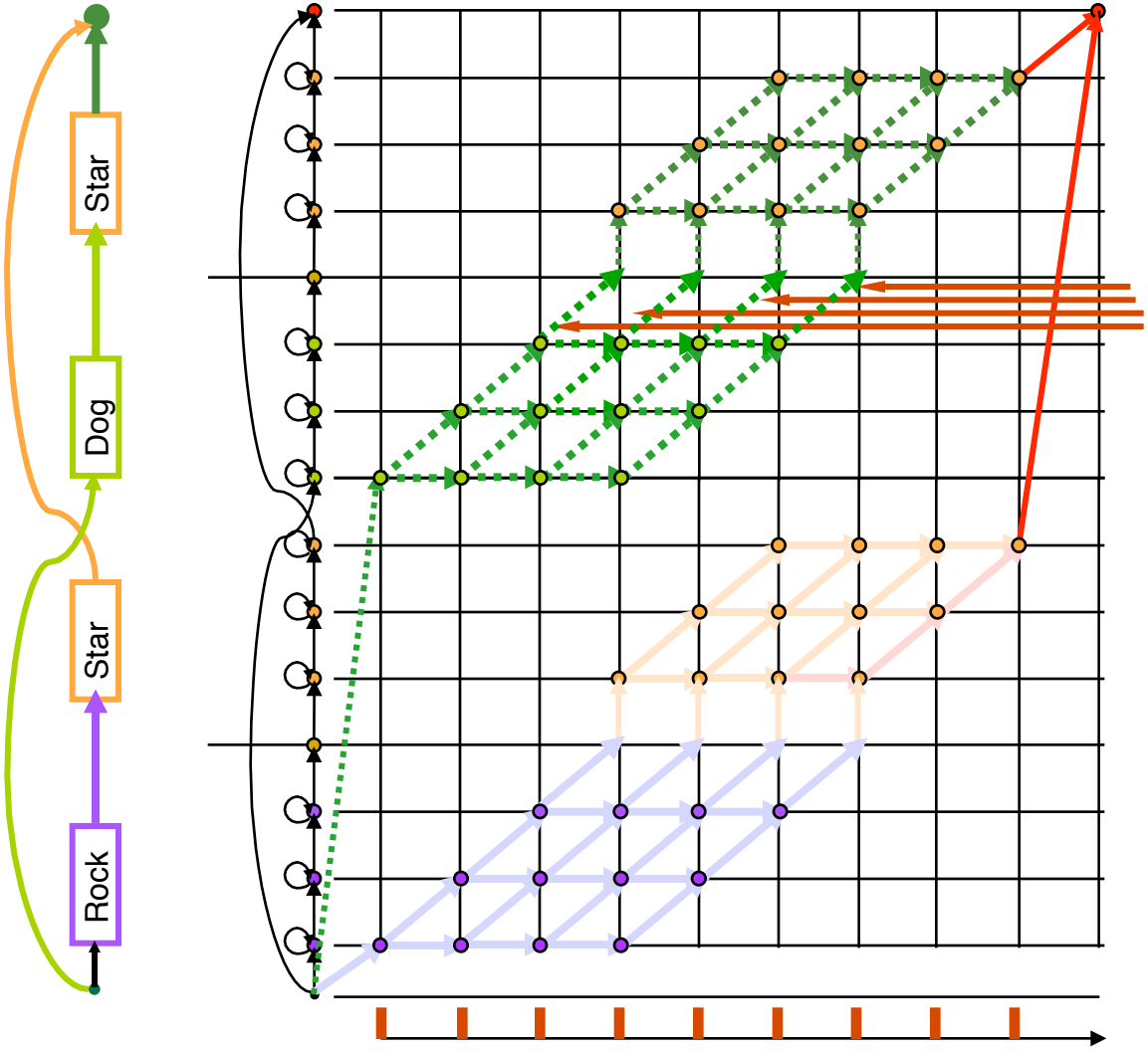


Decoding to classify between word sequences



Approximate total probability with best path score

Decoding to classify between word sequences

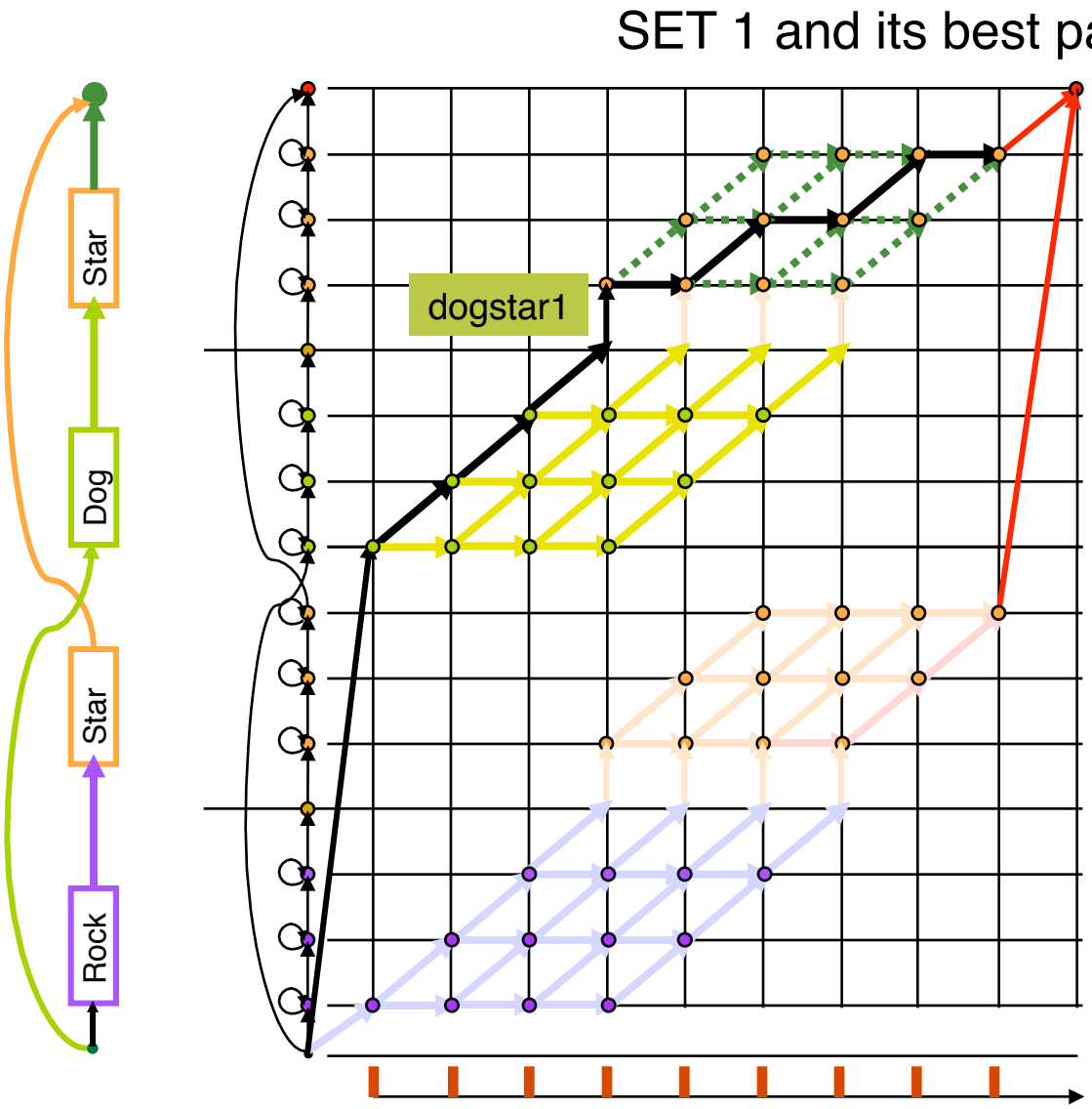


The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from *Dog* to *Star* in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

Decoding to classify between word sequences

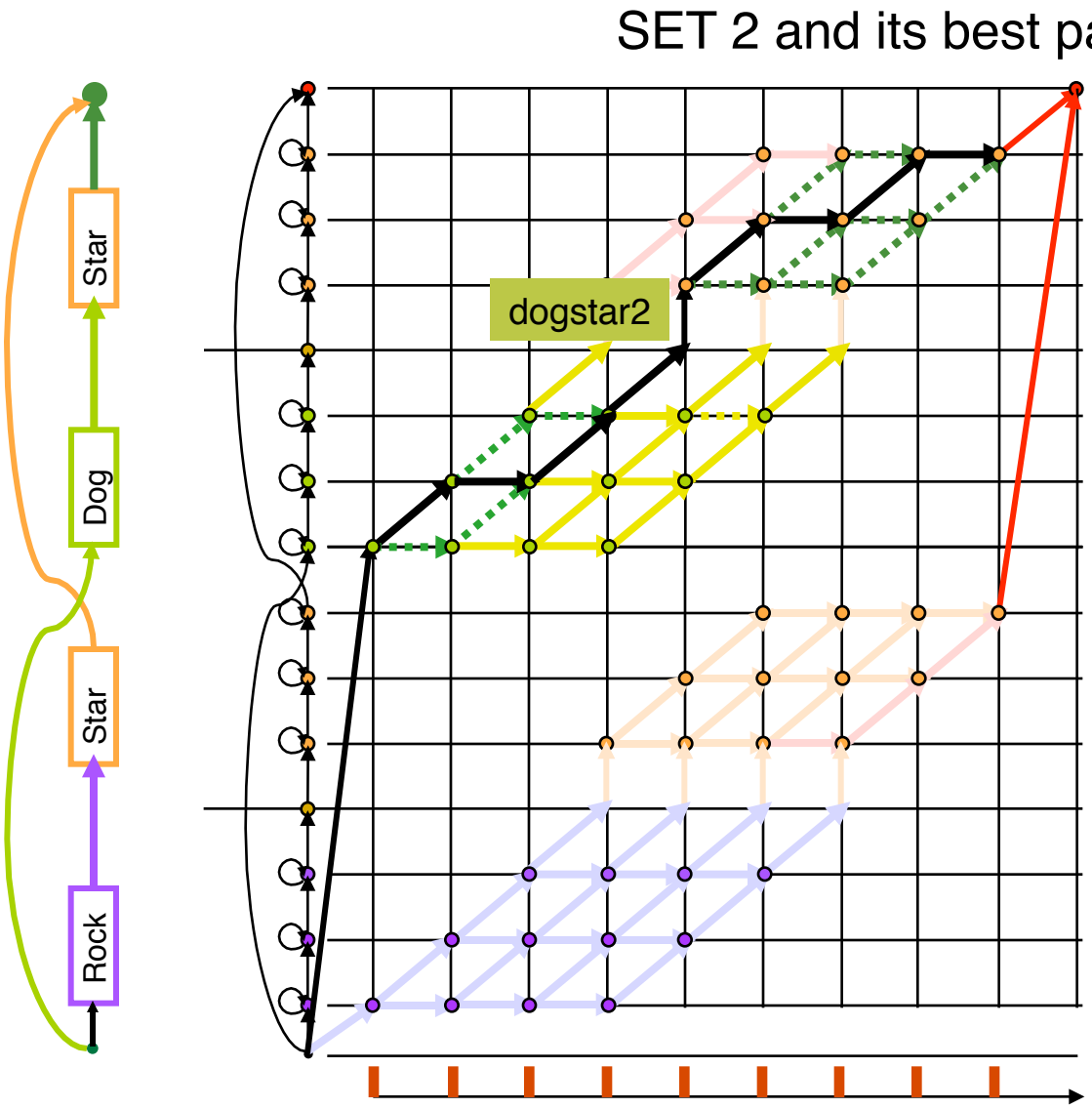


The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from Dog to Star in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

Decoding to classify between word sequences

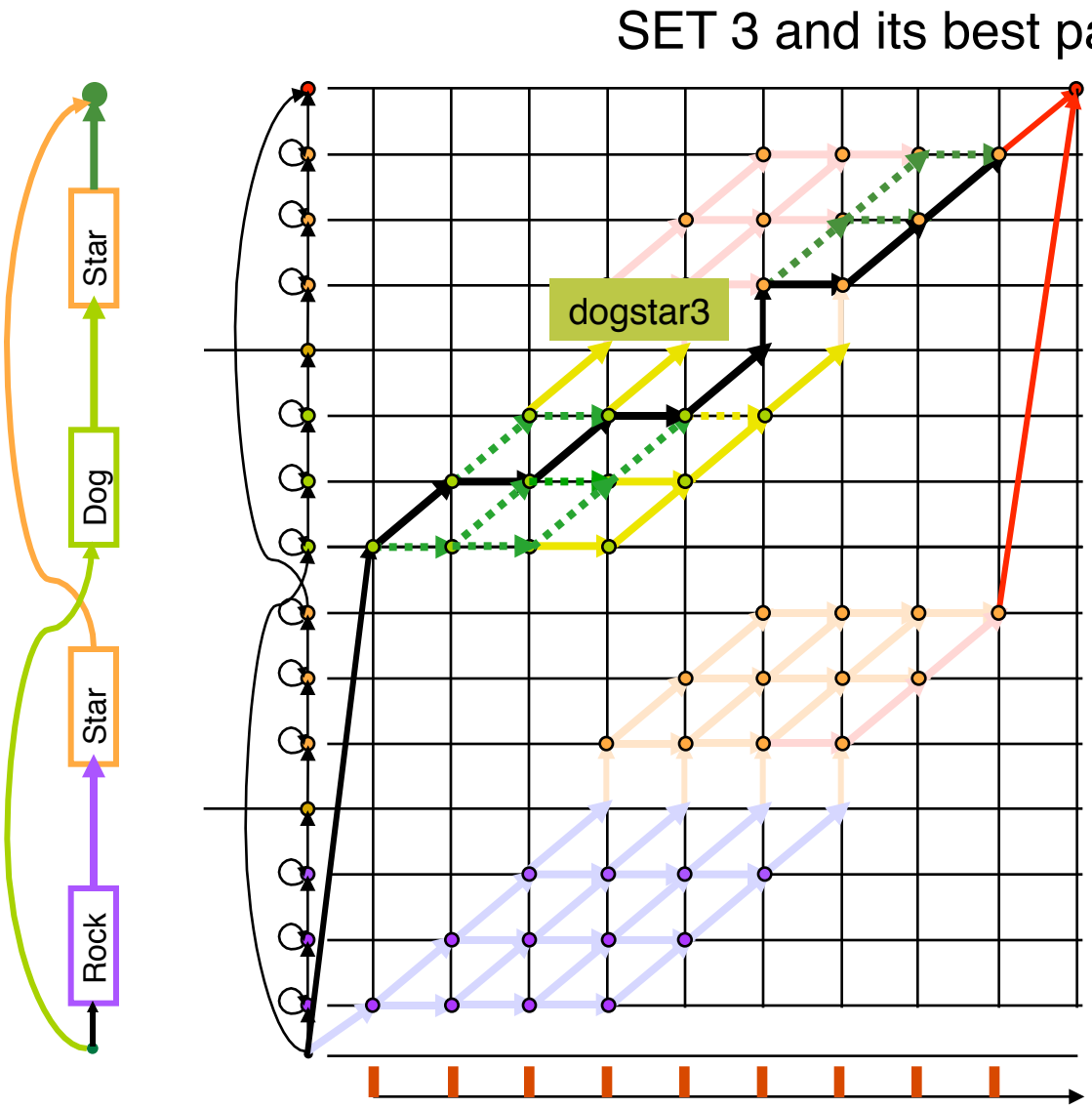


The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from Dog to Star in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

Decoding to classify between word sequences

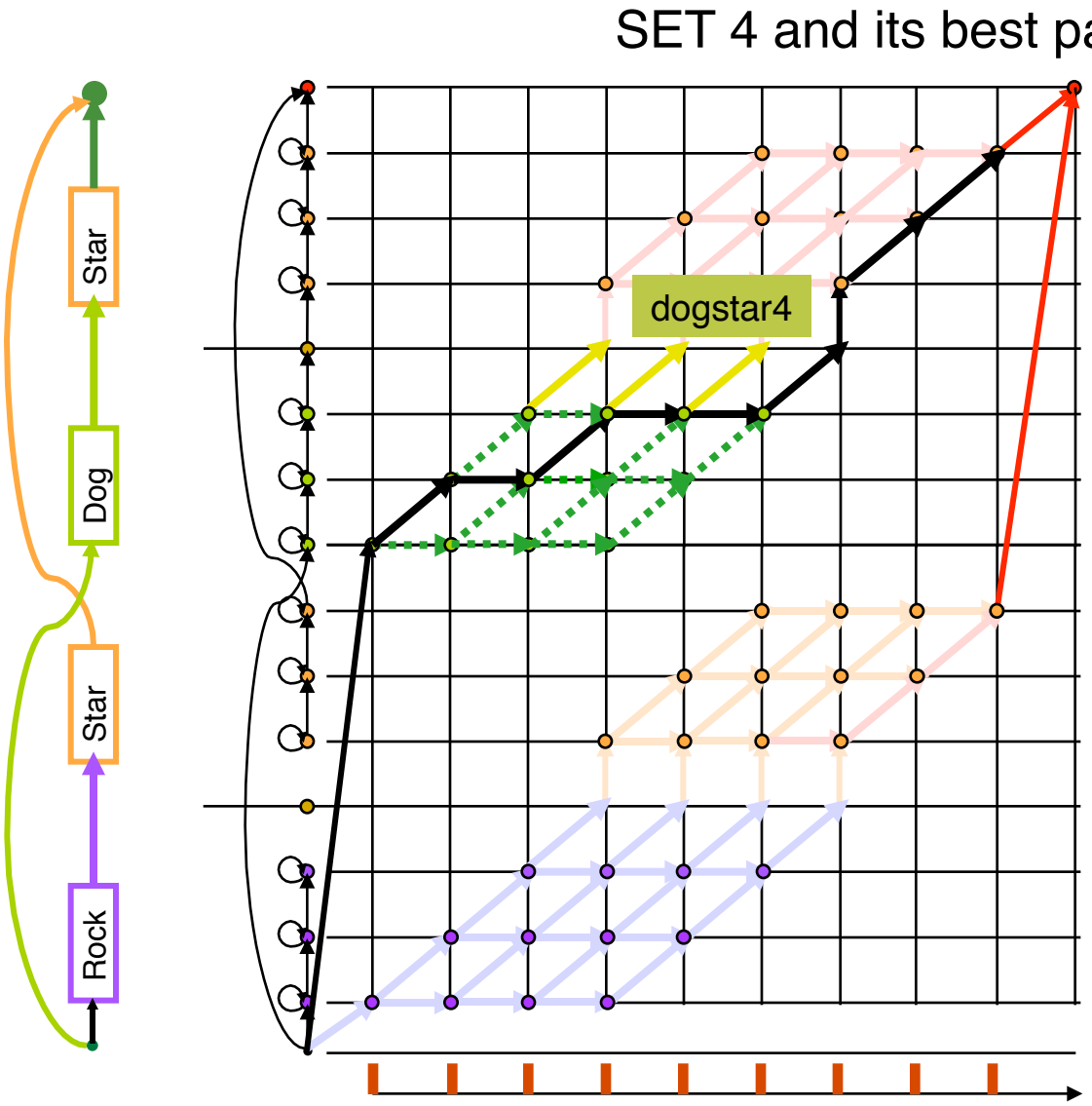


The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from Dog to Star in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

Decoding to classify between word sequences

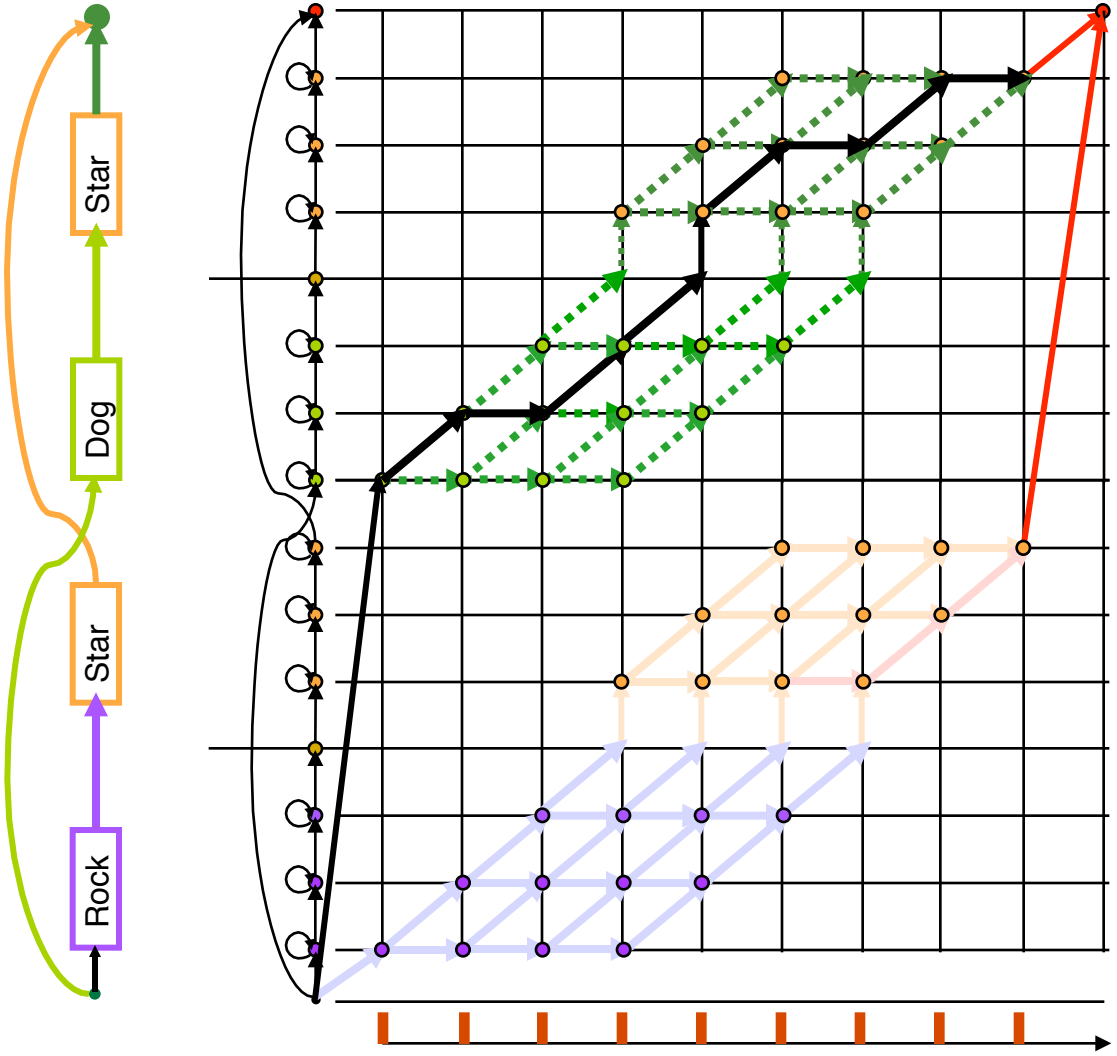


The best path through *Dog Star* lies within the dotted portions of the trellis

There are four transition points from Dog to Star in this trellis

There are four different sets paths through the dotted trellis, each with its own best path

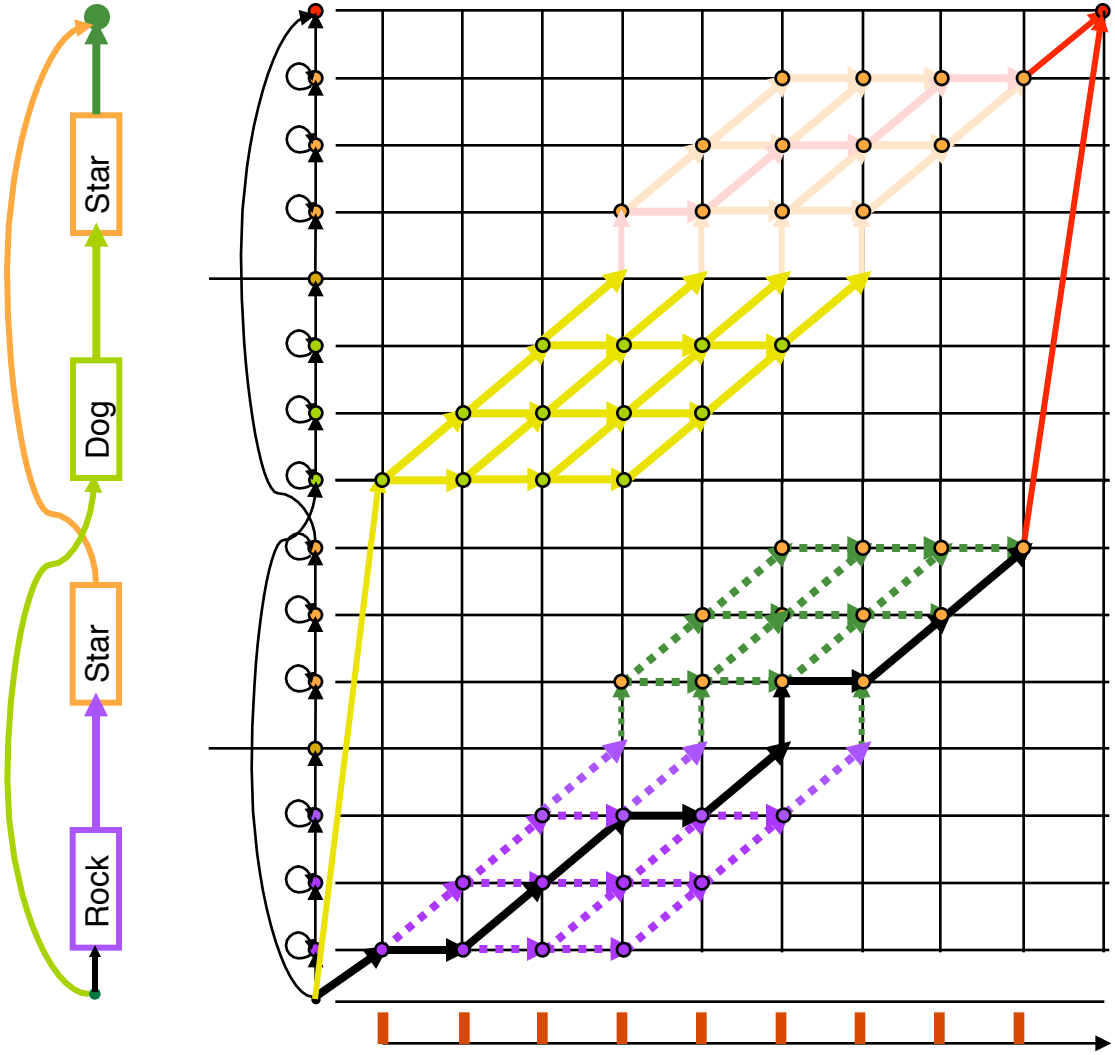
Decoding to classify between word sequences



The best path through *Dog Star* is the best of the four transition-specific best paths

max(dogstar) =
max (dogstar1, dogstar2,
dogstar3, dogstar4)

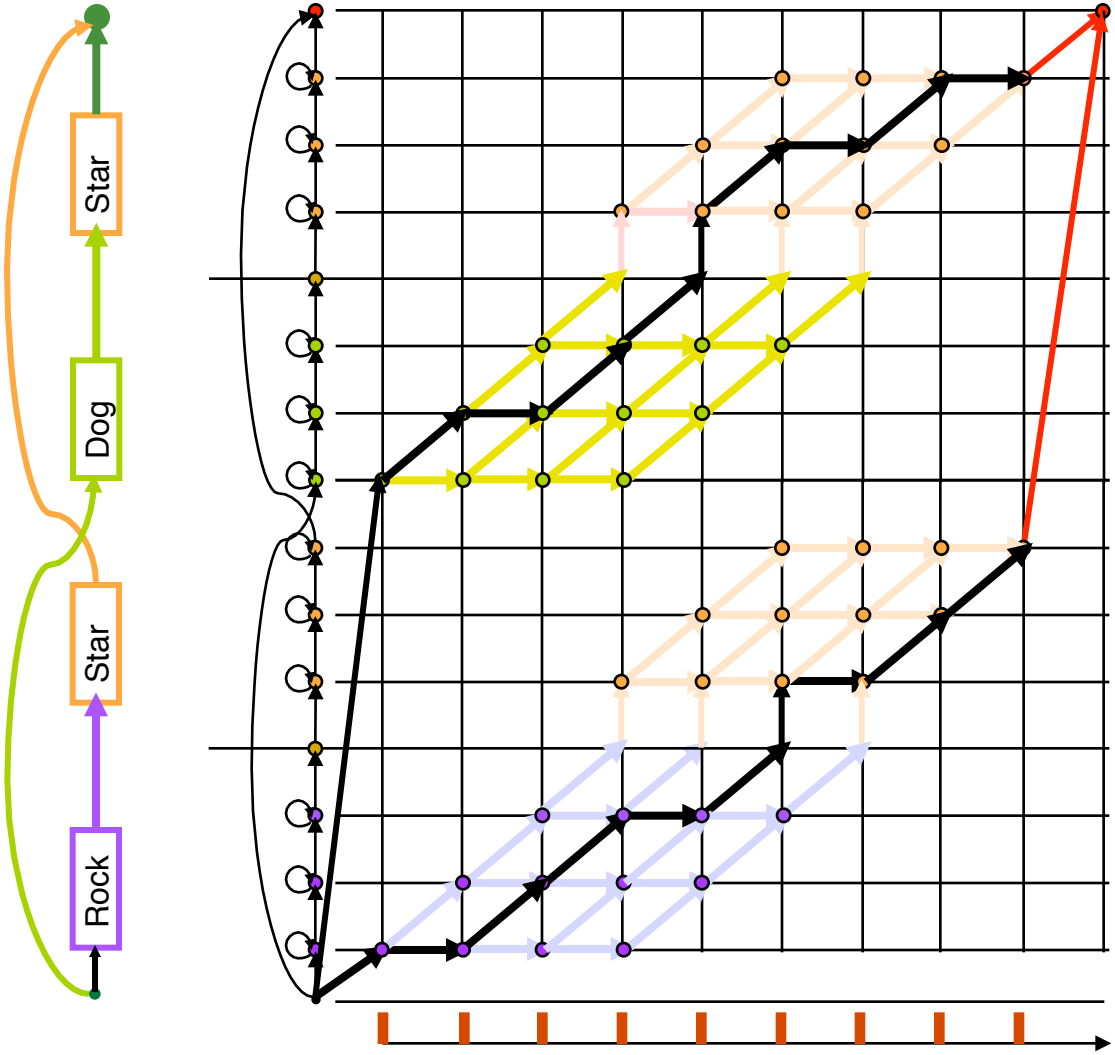
Decoding to classify between word sequences



Similarly, for *Rock Star* the best path through the trellis is the best of the four transition-specific best paths

max(rockstar) =
max (rockstar1, rockstar2,
rockstar3, rockstar4)

Decoding to classify between word sequences



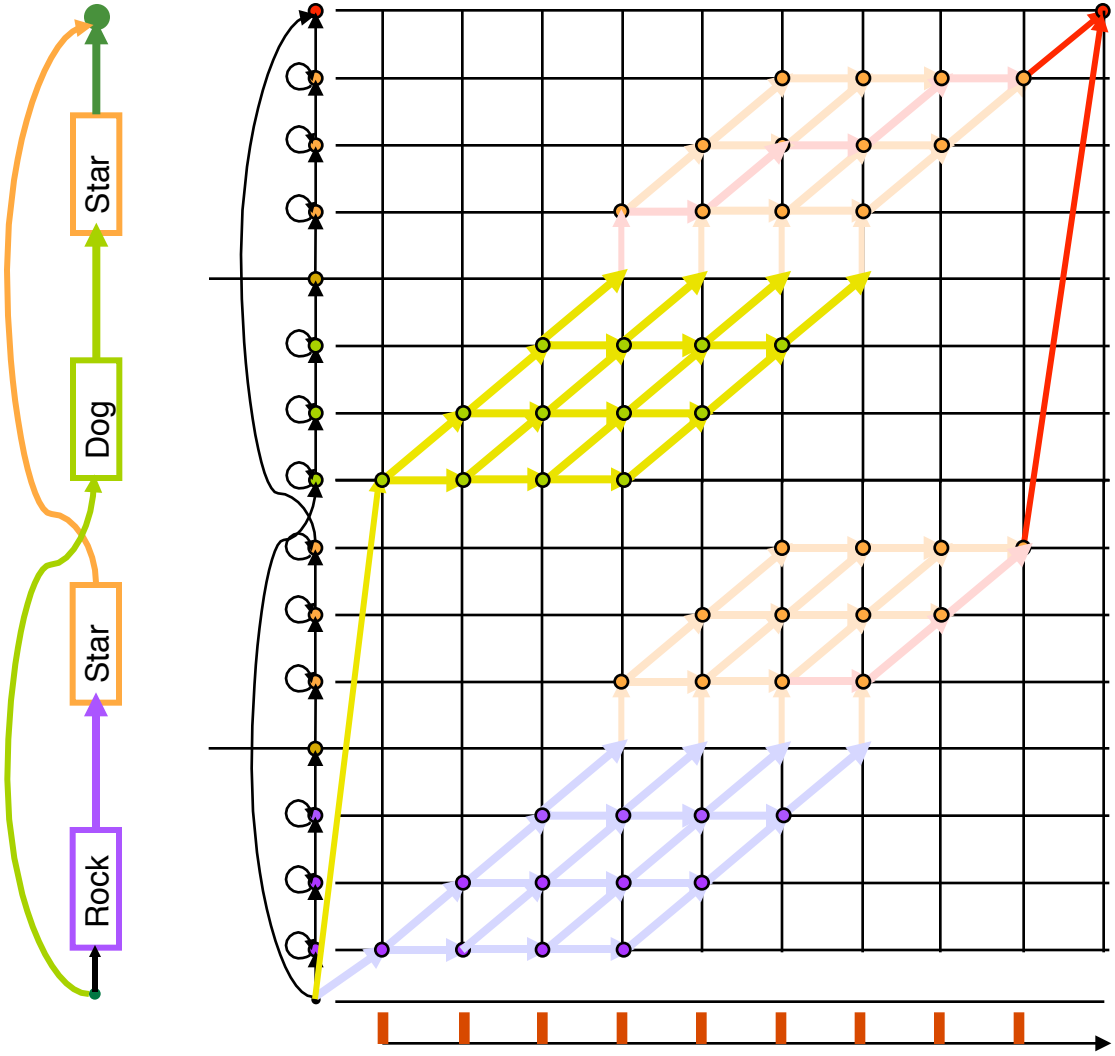
Then we'd compare the best paths through *Dog Star* and *Rock Star*

max(dogstar) =
max (dogstar1, dogstar2,
dogstar3, dogstar4)

max(rockstar) =
max (rockstar1, rockstar2,
rockstar3, rockstar4)

Viterbi =
max(max(dogstar),
max(rockstar))

Decoding to classify between word sequences



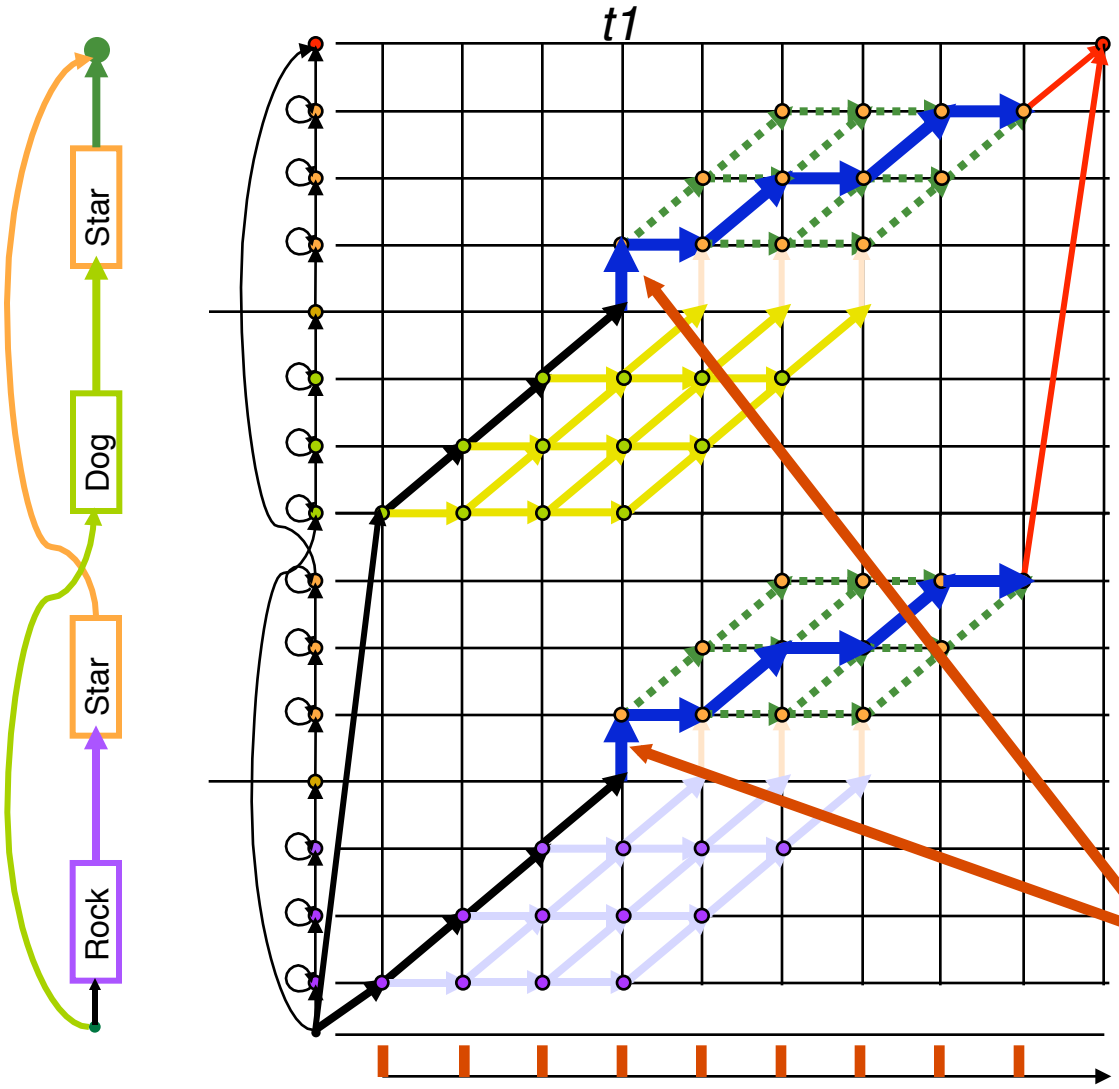
argmax is commutative:

max(max(dogstar),
max(rockstar))

=

max (
  max(dogstar1, rockstar1),
  max (dogstar2, rockstar2),
   max (dogstar3,rockstar3),
   max(dogstar4,rockstar4)
)

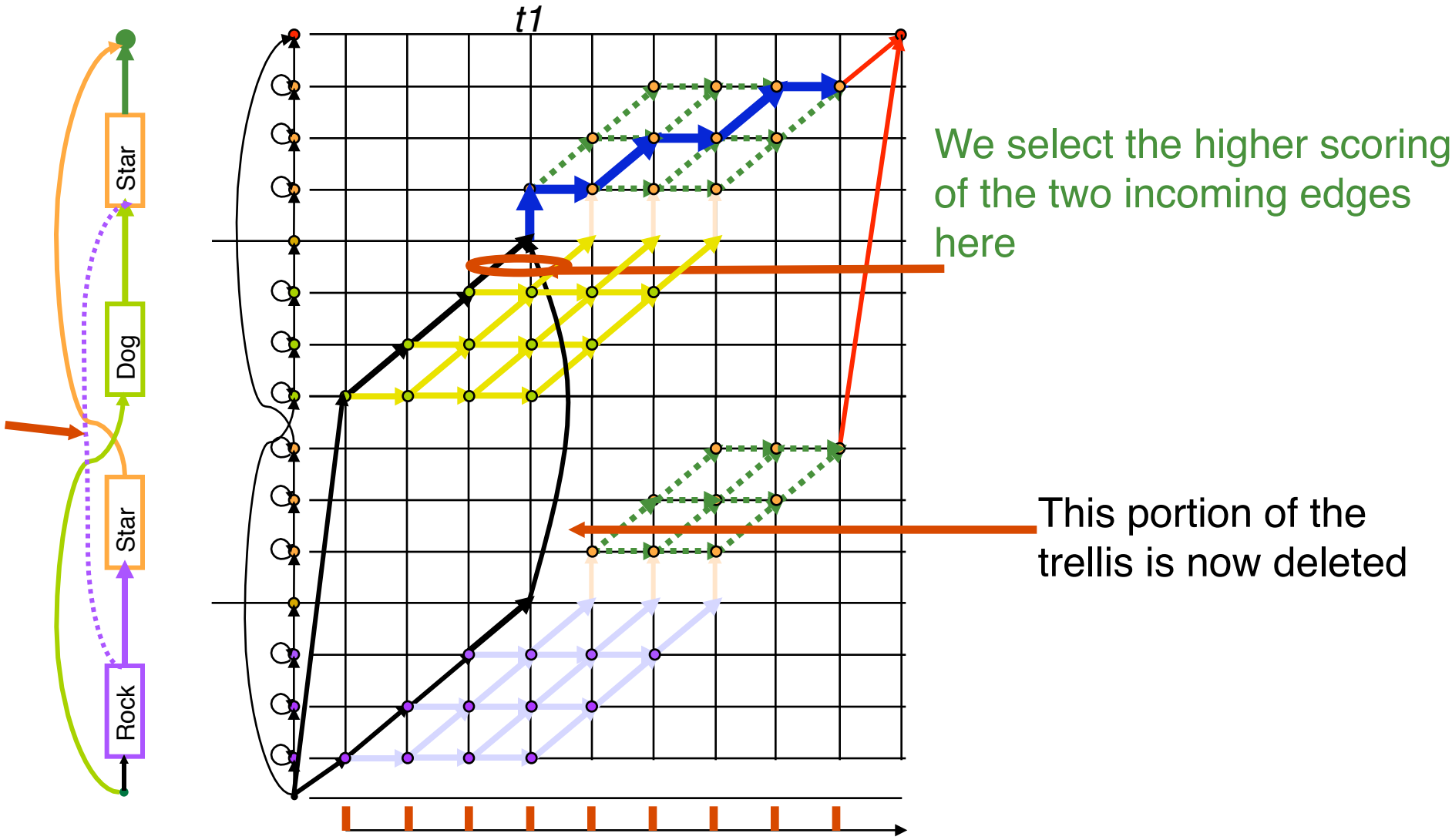
Decoding to classify between word sequences



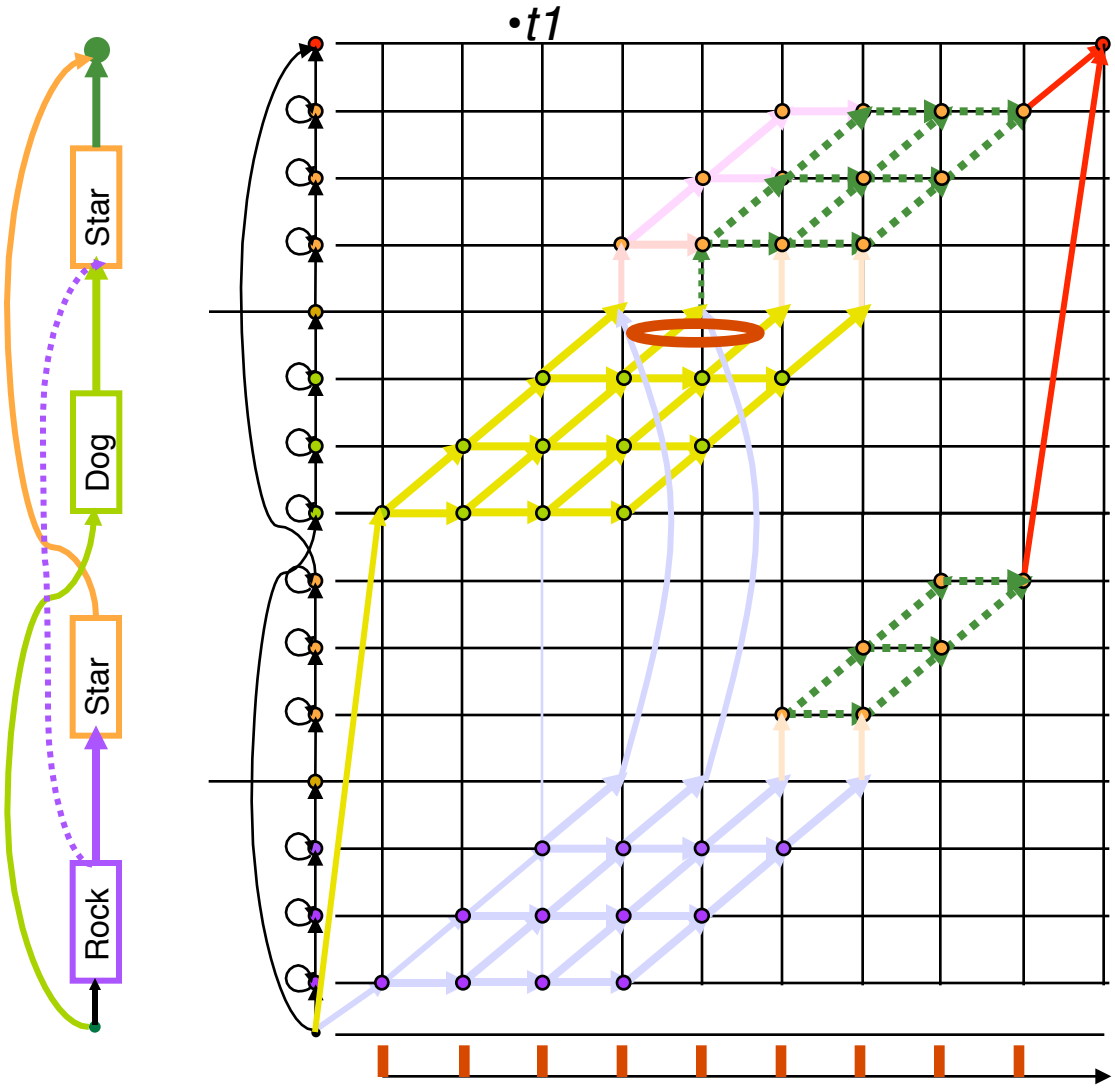
For a given entry point the best path through STAR is the same for both trellises

We can choose between Dog and Rock right here because the futures of these paths are identical

Decoding to classify between word sequences

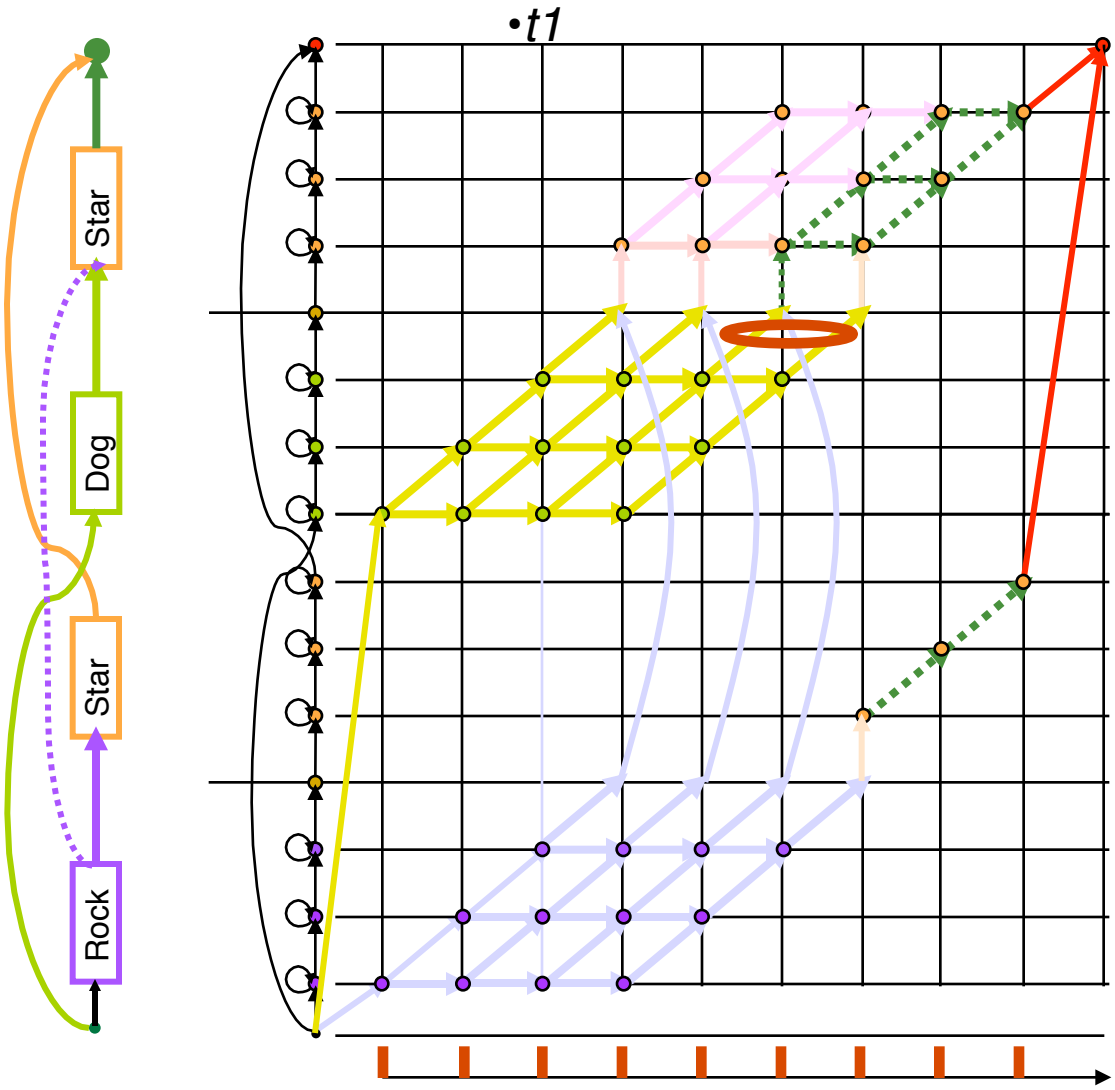


Decoding to classify between word sequences



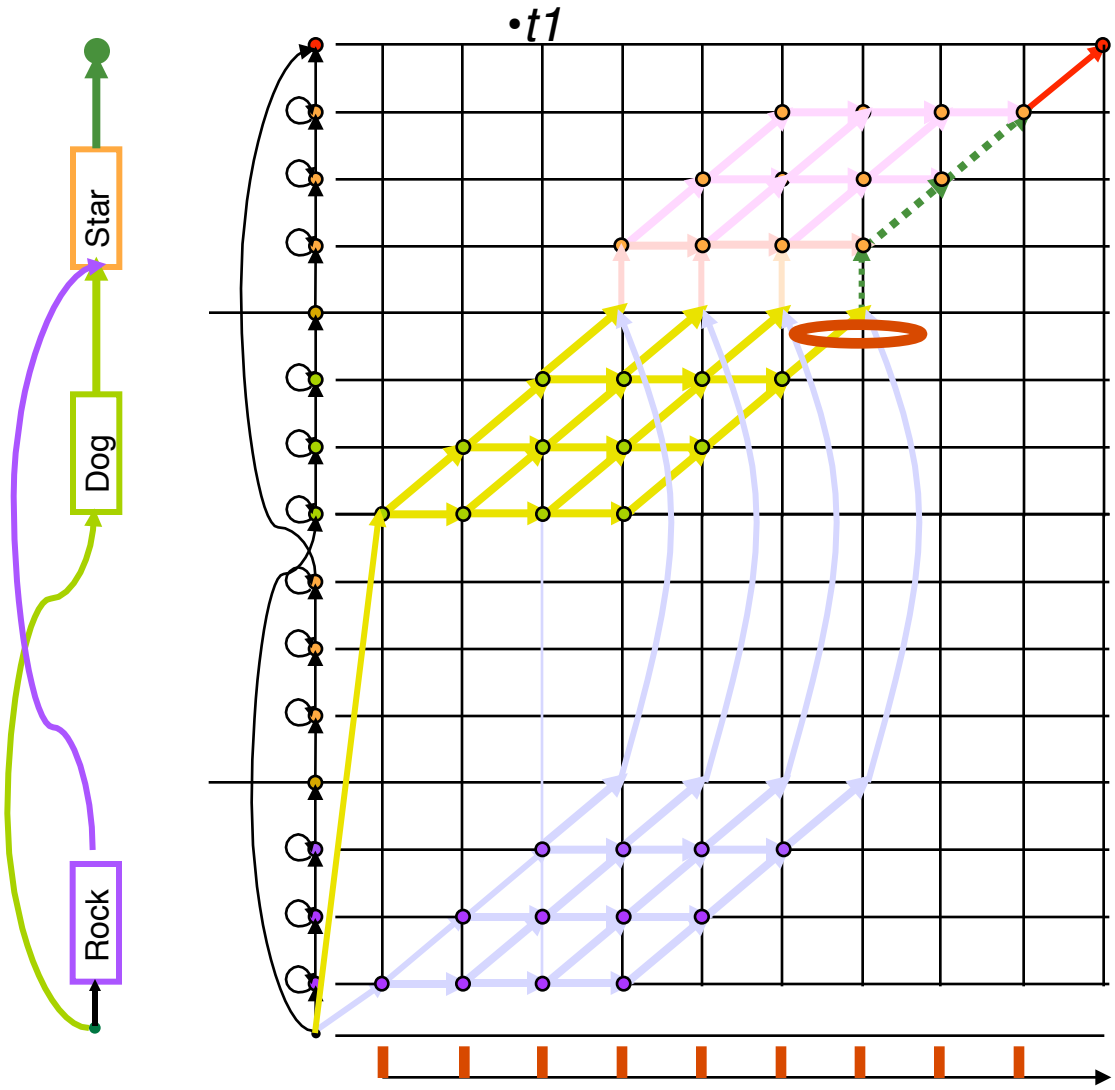
Similar logic can be applied at other entry points to *Star*

Decoding to classify between word sequences



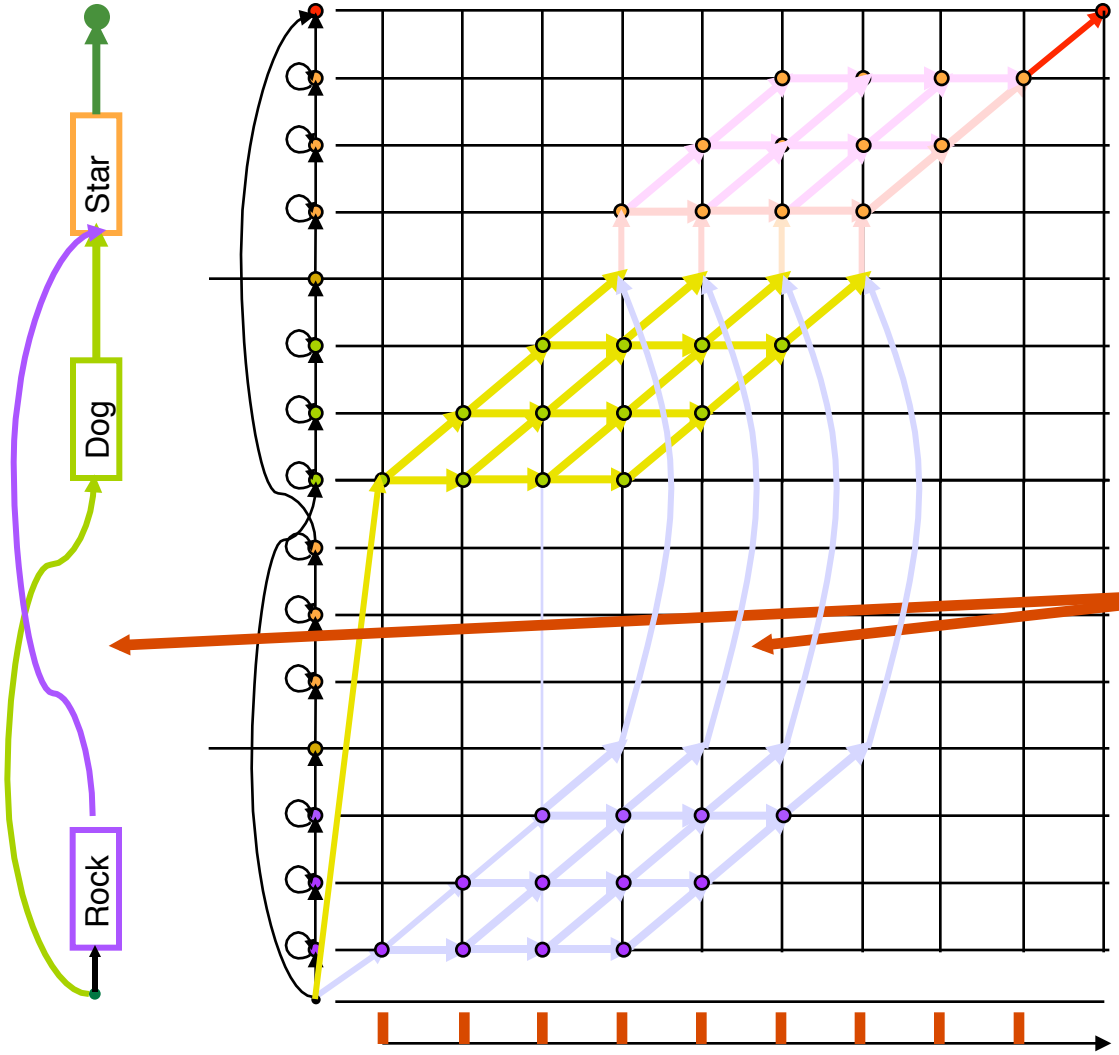
Similar logic can be applied at other entry points to *Star*

Decoding to classify between word sequences



Similar logic can be applied at other entry points to *Star*

Decoding to classify between word sequences

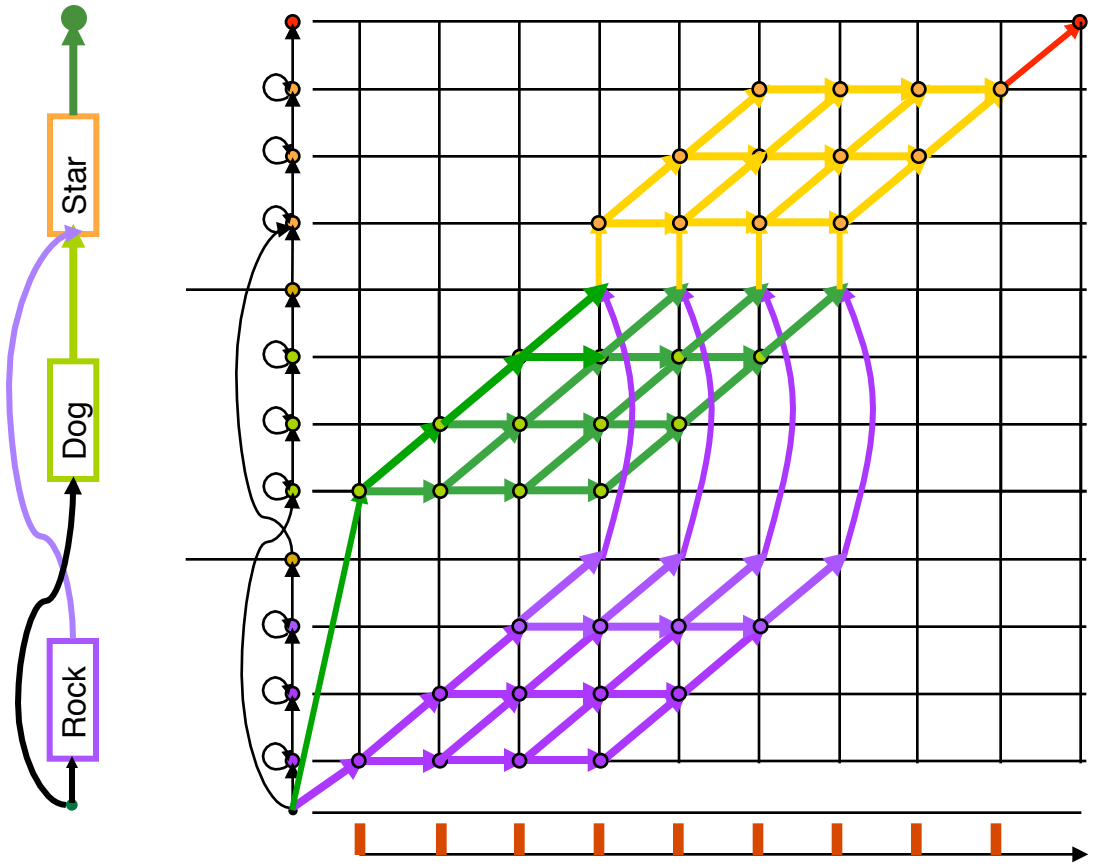


Similar logic can be applied at other entry points to *Star*

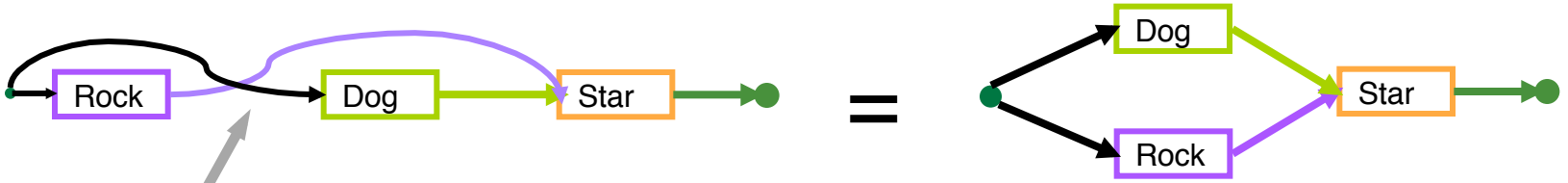
This copy of the trellis for *STAR* is completely removed

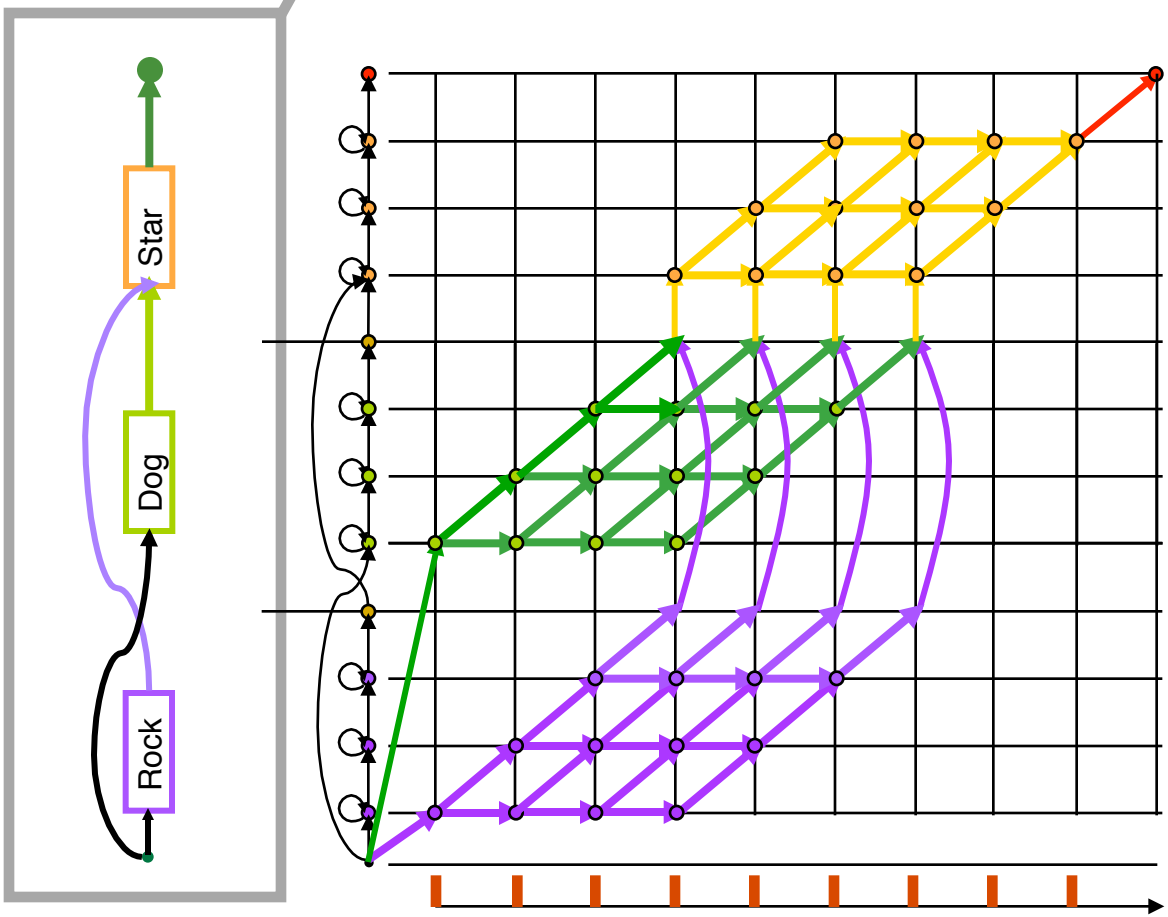
Decoding to classify between word sequences

- The two instances of *Star* can be collapsed into one to form a smaller trellis



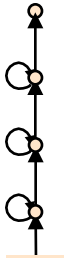
Language-HMMs for fixed length word sequences



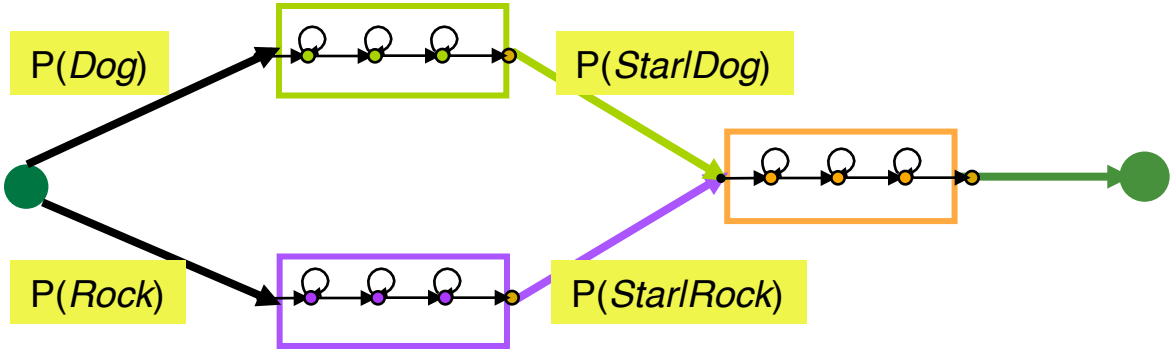


We will represent the vertical axis of the trellis in this simplified manner

Language-HMMs for fixed length word sequences

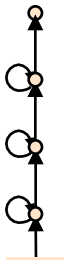


Each word is an HMM



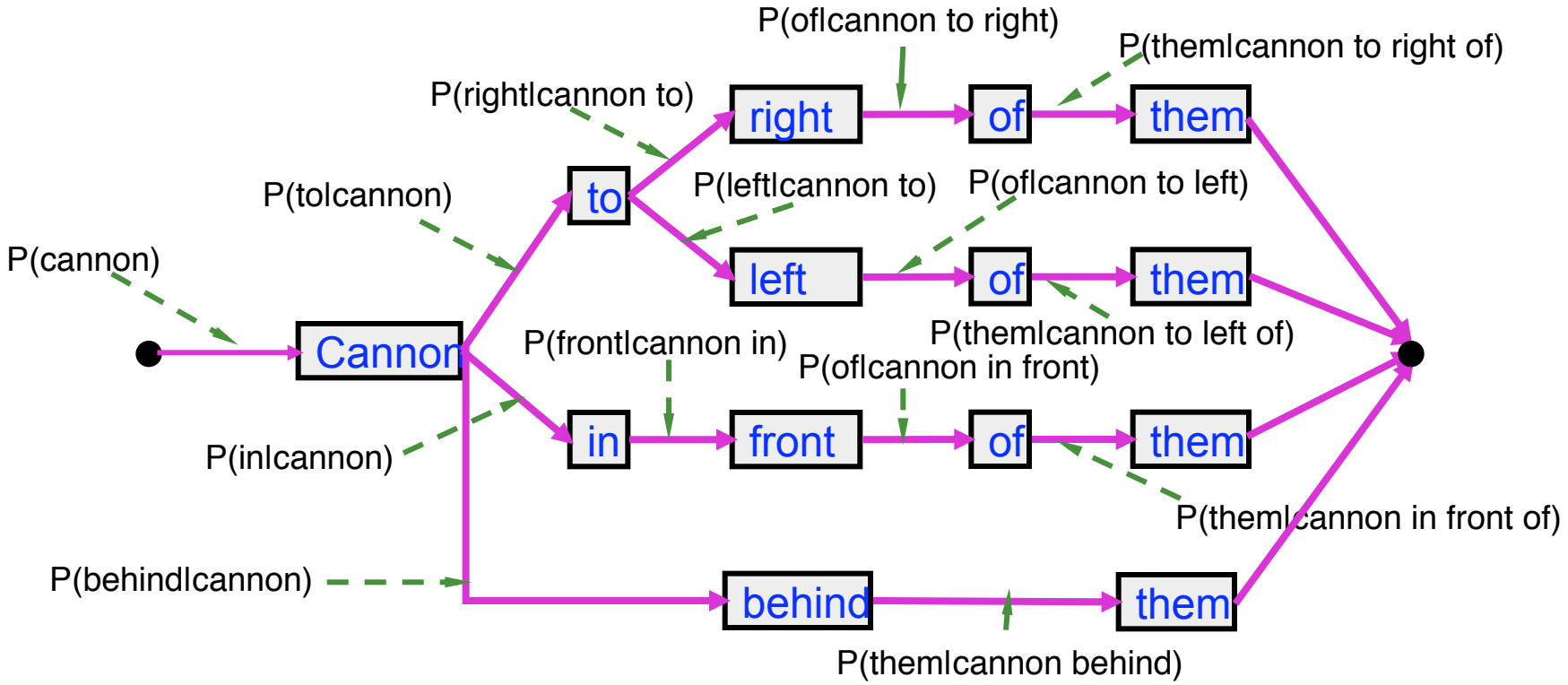
- The word graph represents all allowed word sequences in our example
  - The set of all allowed word sequences represents the allowed "language"
- At a more detailed level, the figure represents an HMM composed of the HMMs for all words in the word graph
  - This is the "Language HMM" – the HMM for the entire allowed language
- The language HMM represents the vertical axis of the trellis
  - It is the trellis, and NOT the language HMM, that is searched for the best path

Language-HMMs for fixed length word sequences



Each word is an HMM

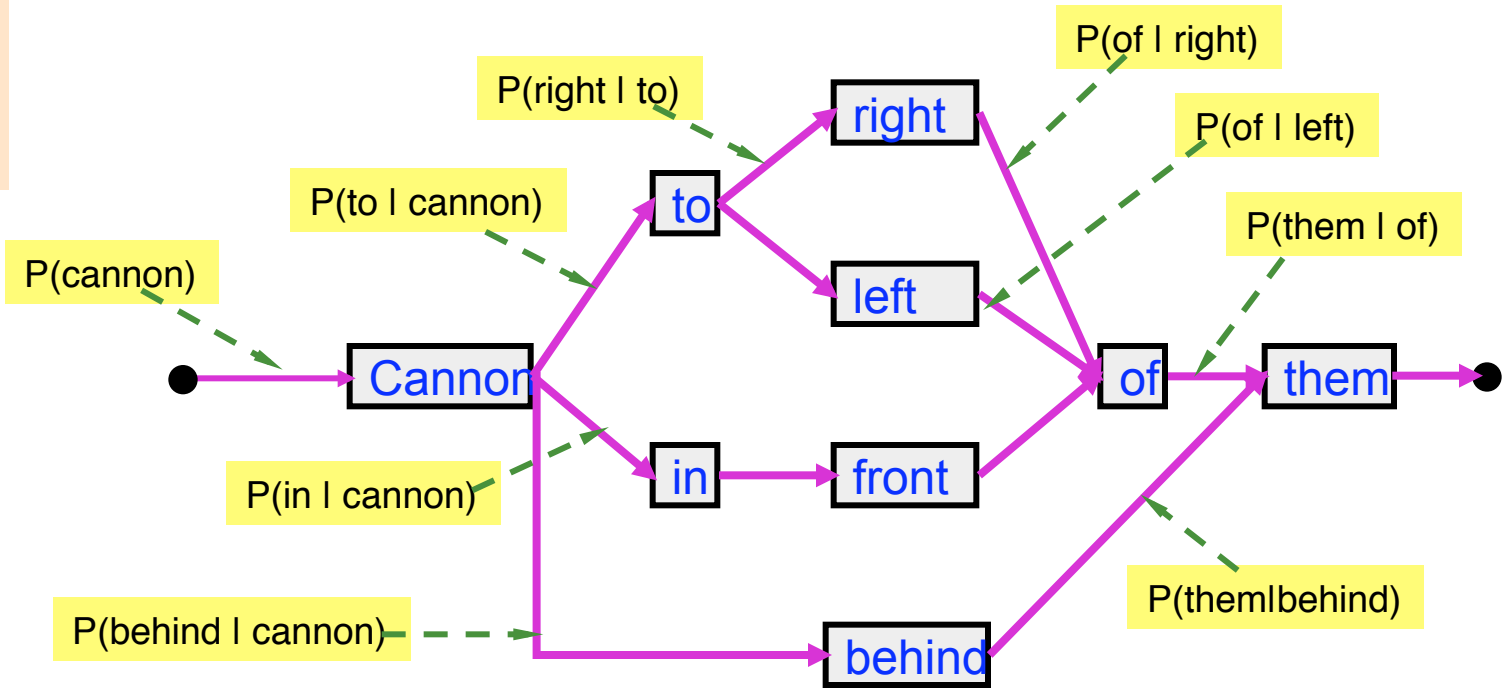
- Recognizing one of four lines from "charge of the light brigade"
  - Cannon to right of them
  - Cannon to left of them
  - Cannon in front of them
  - Cannon behind them



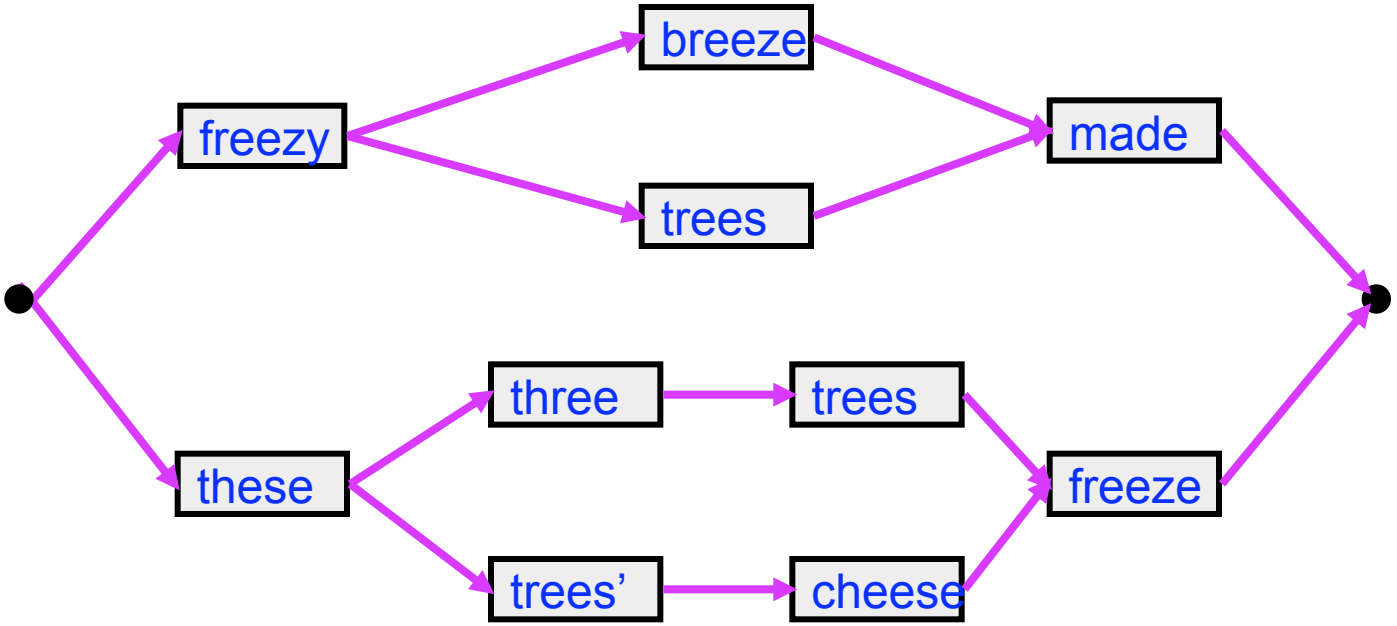
Simplification of the language HMM through lower context language models

Each word is an HMM

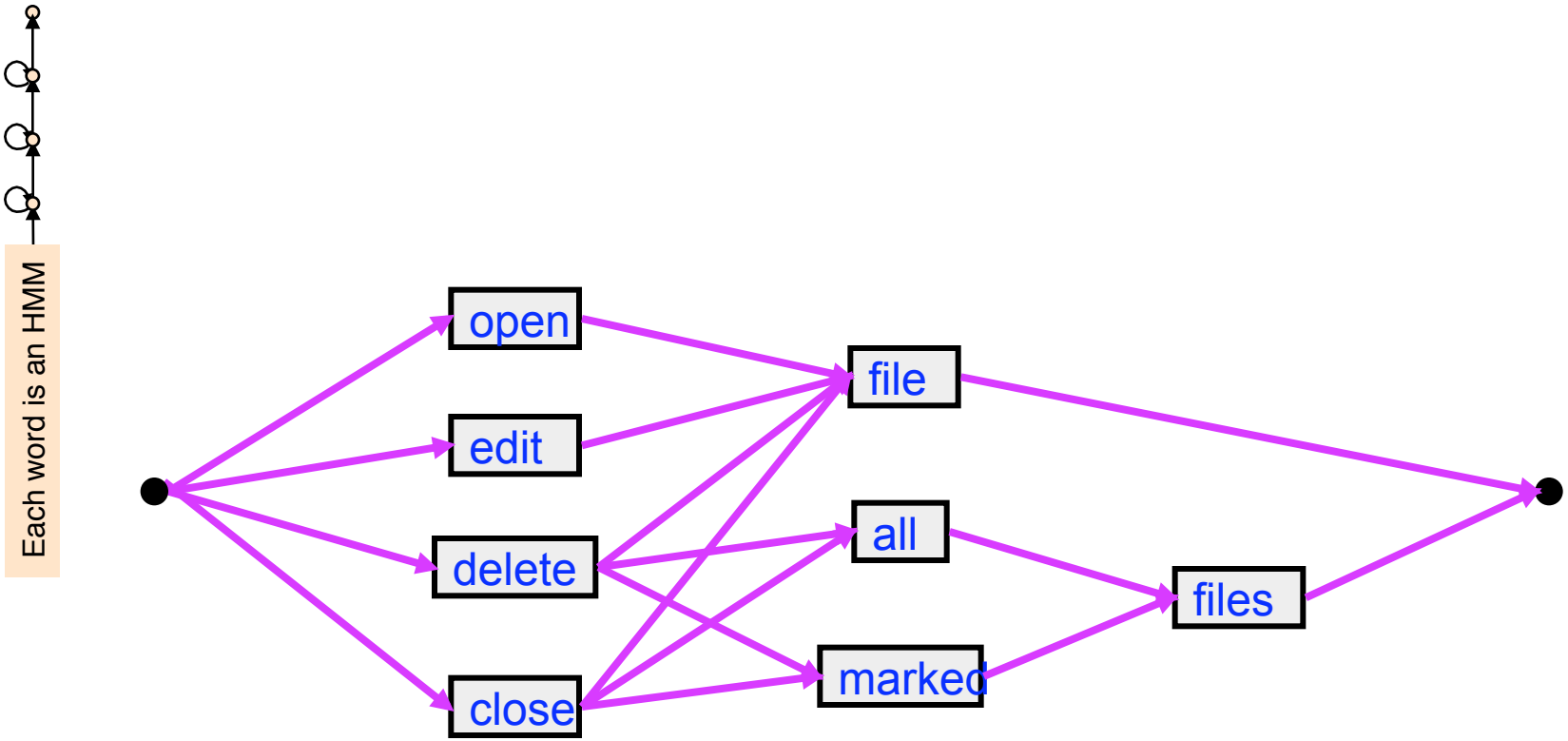
- Recognizing one of four lines from “charge of the light brigade”
- If the probability of a word only depends on the preceding word, the graph can be collapsed:
  - **e.g. P(them | cannon to right of) = P(them | cannon to left of) = P(cannon | of)**



Language HMMs for fixed-length word sequences: based on a grammar for Dr. Seuss



Language HMMs for fixed-length word sequences: command and control grammar



Language HMMs for arbitrarily long word sequences

- Constrained set of word sequences with constrained vocabulary are realistic
  - Typically in command-and-control situations
    - Example: operating TV remote
  - Simple dialog systems
    - When the set of permitted responses to a query is restricted
- Unconstrained word sequences : NATURAL LANGUAGE
  - State-of-art large vocabulary decoders

Language HMMs for natural language: N-gram representations

- Unigram Model: A bag of words model:
  - The probability of a word is independent of the words preceding or succeeding it.

P(*When you wish upon a star*) =

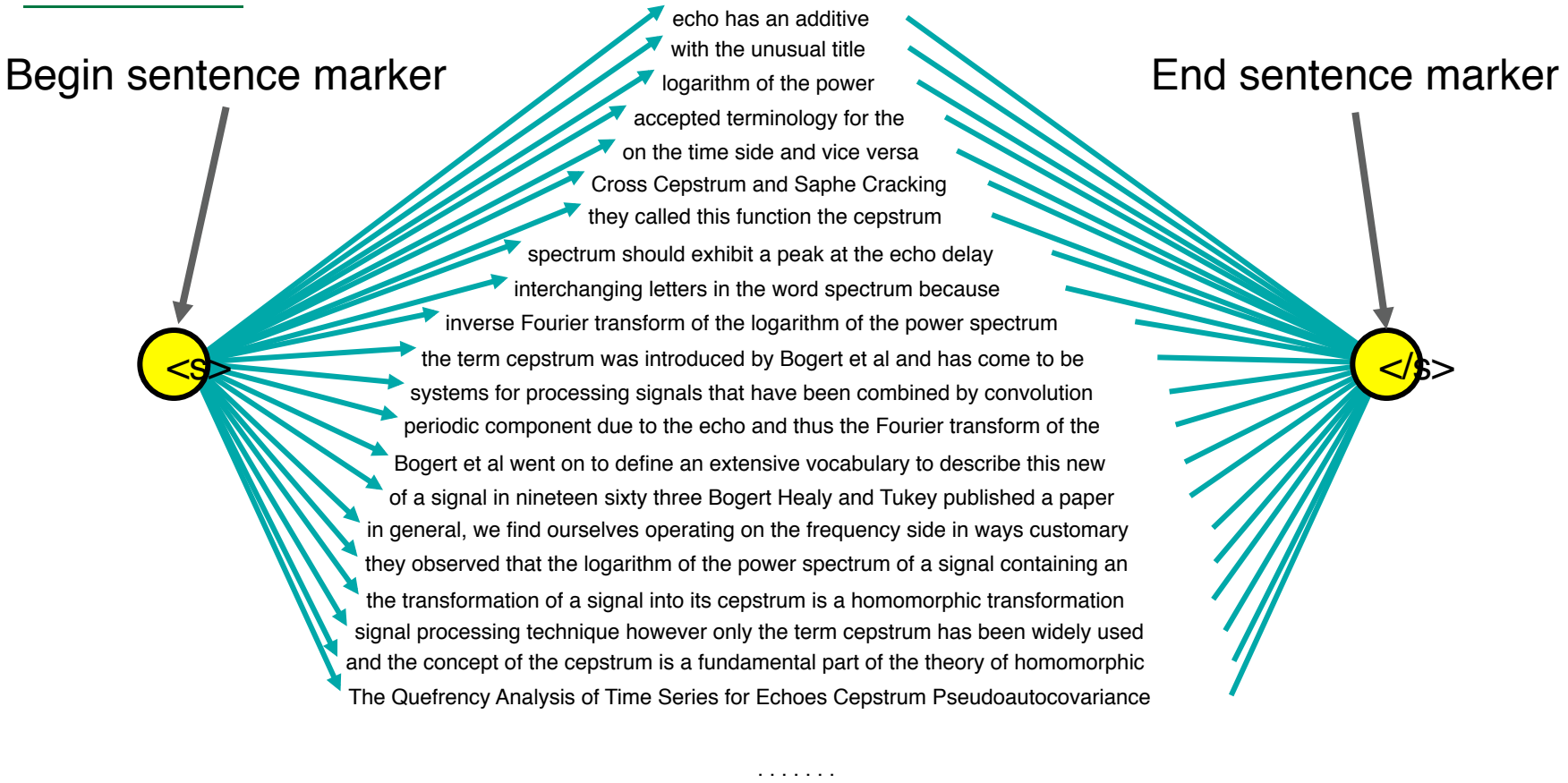
P(*When*) *P*(*you*) *P*(*wish*) *P*(*upon*) *P*(*a*) *P*(*star*) *P*(END)

- "END" is a special symbol, that indicates the end of the word sequence
  - P(END) is necessary – without it the word sequence would never terminate

Language HMMs for Natural language: N-Gramrepresentations

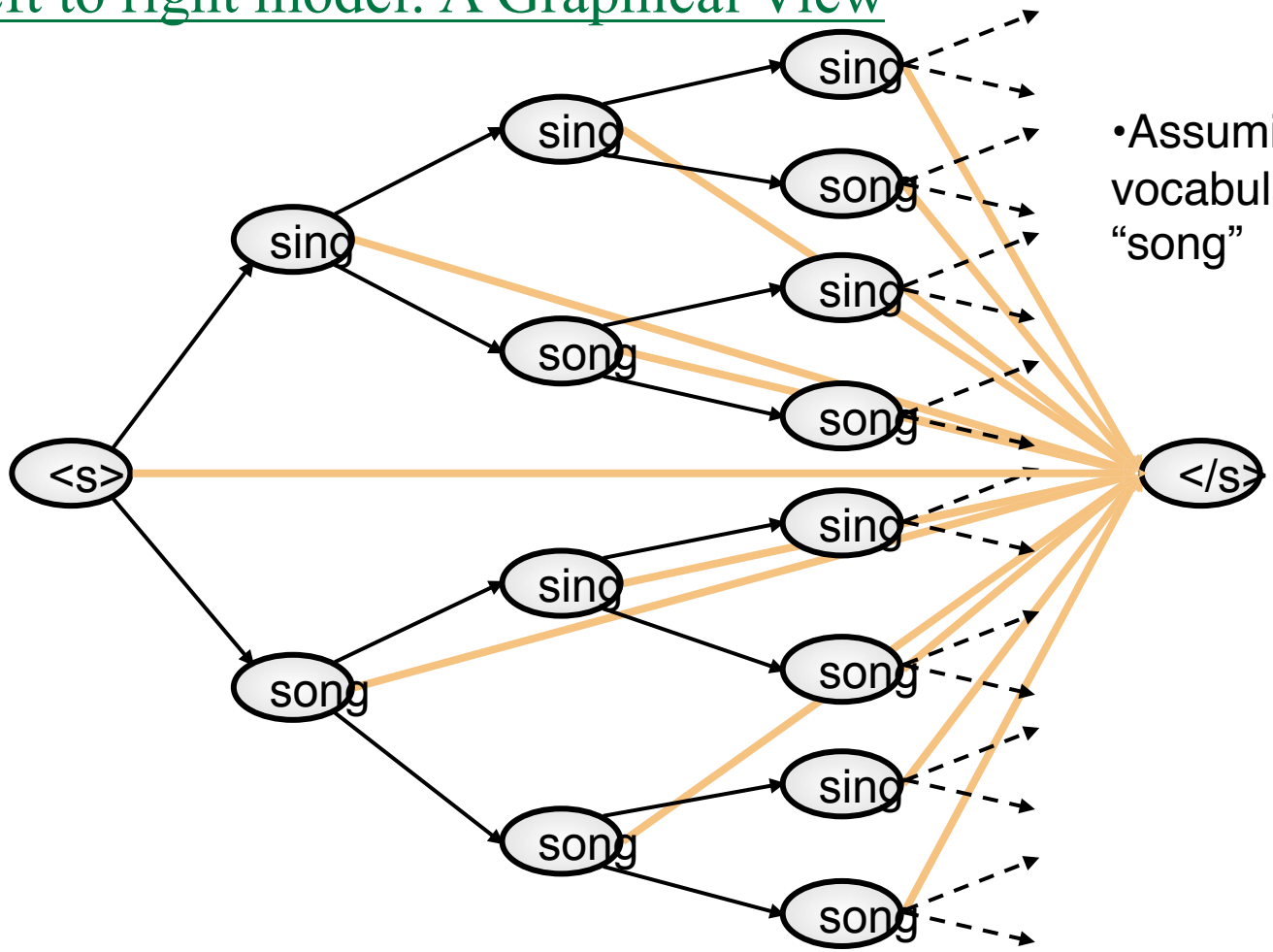
- Bigram language model: the probability of a word depends on the previous word
  - P(When you wish upon a star) = P(When|START) P(you |when) P(wish |you) …. P(Star | a) P(END|Star)
- Trigram representations
  - P(When you wish upon a star) = P(When|START) P(you |START when) P(wish |when you) …. P(Star |upon a) P(END|a Star)
- Ngram representations allow us to represent free-form language as finite graphs

Recognizing Natural Language: Choose between all infinite sentences



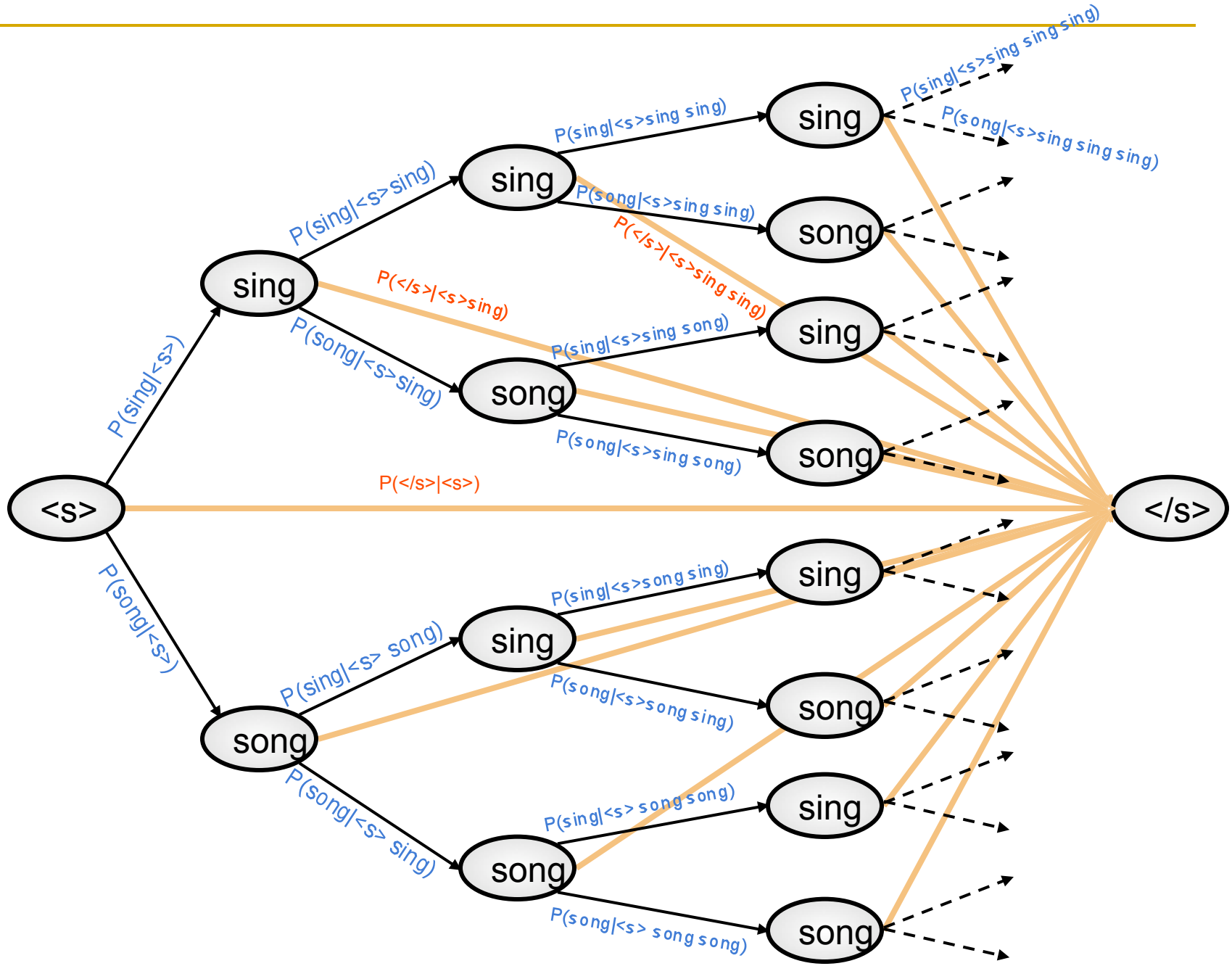
- There will be one path for every possible word sequence
- A priori probabilitiy for a word sequence can be applied anywhere along the path representing that word sequence.
- It is the structure and size of this graph that determines the feasibility of the recognition task

The left to right model: A Graphical View

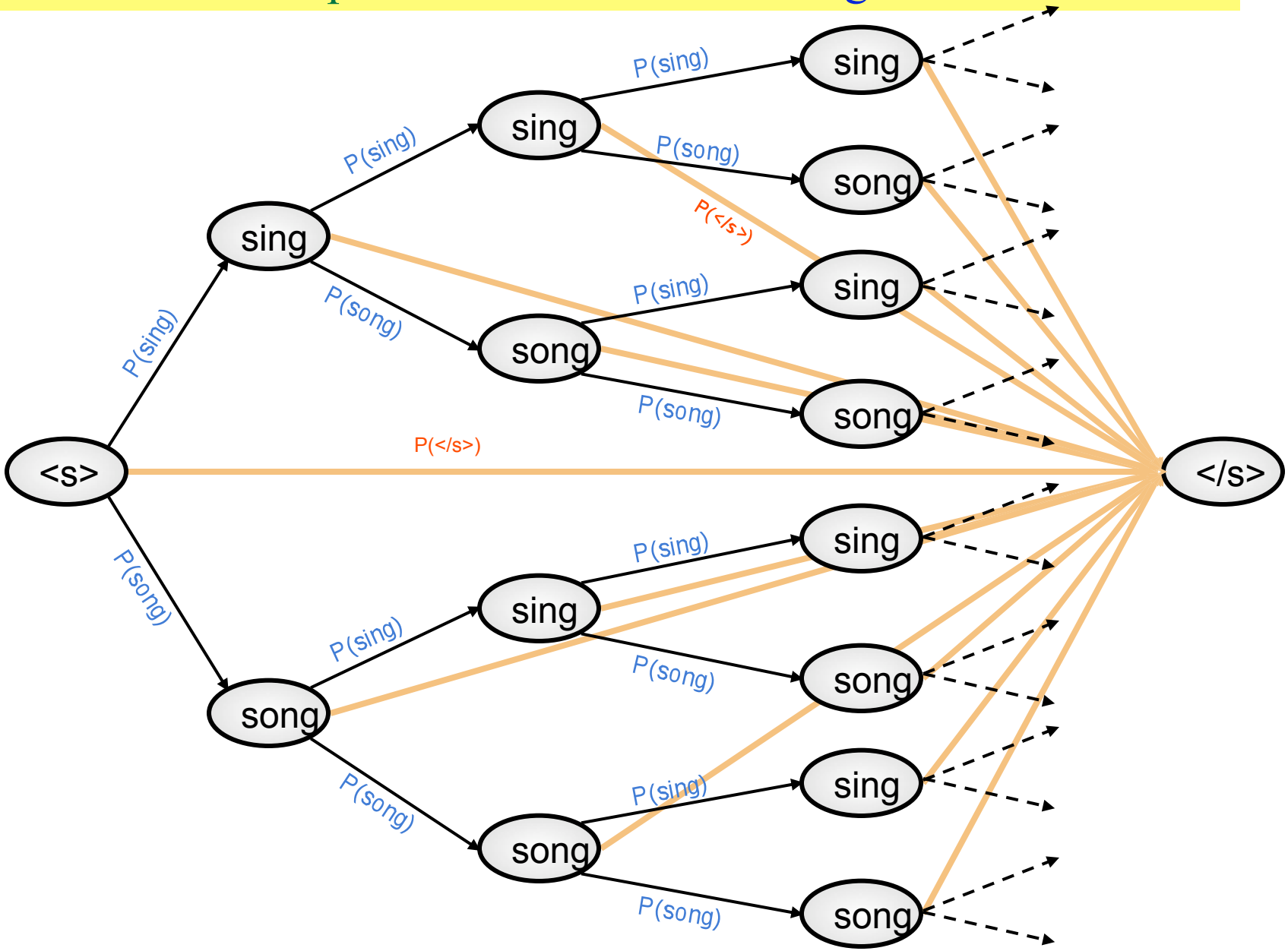


•Assuming a two-word vocabulary: “sing” and “song”

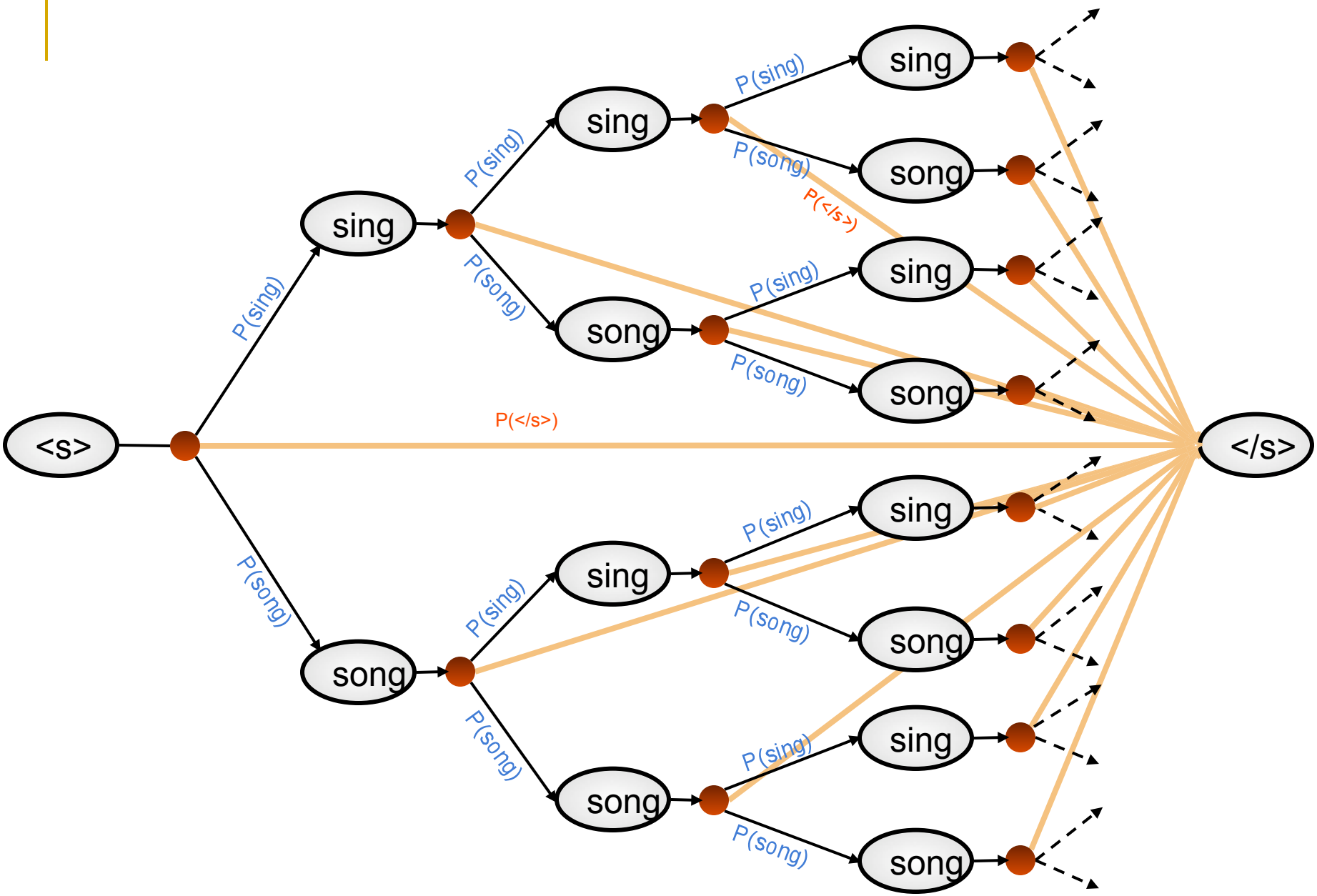
- *A priori* probabilities for word sequences are spread through the graph
  - They are applied on every edge
- This is a much more compact representation of the language than the full graph shown earlier
  - But is still inifinitely large in size

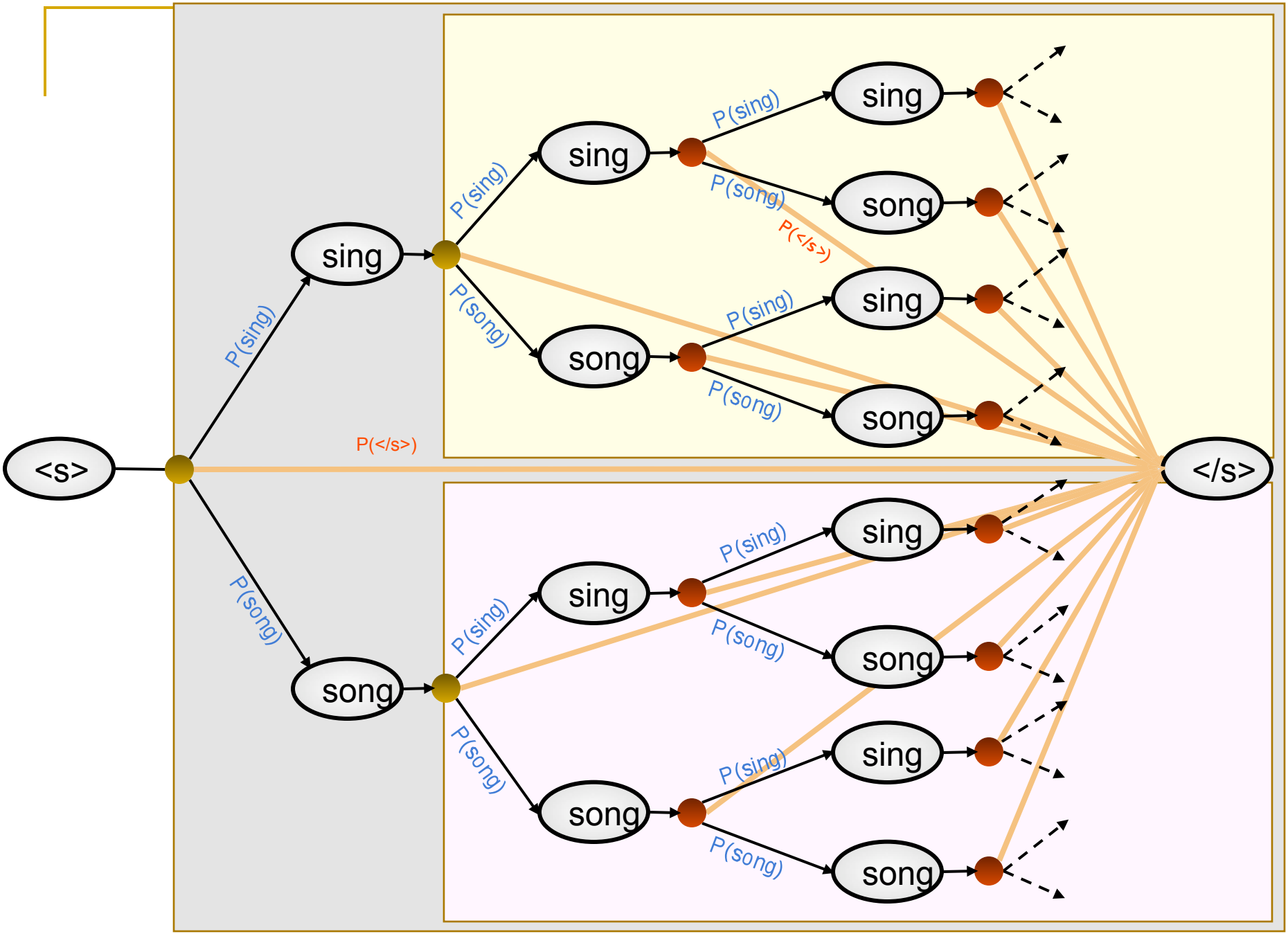

<s>
sing
song
</s>
P(sing|<s>)
P(song|<s>)
P(</s>|<s>)
P(sing|<s>sing)
P(song|<s>sing)
P(</s>|<s>sing)
P(sing|<s>song)
P(song|<s>song)
P(</s>|<s>sing sing)
P(sing|<s>sing sing)
P(song|<s>sing sing)
P(sing|<s>sing song)
P(song|<s>sing song)
P(sing|<s>song sing)
P(song|<s>song sing)
P(sing|<s>song song)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

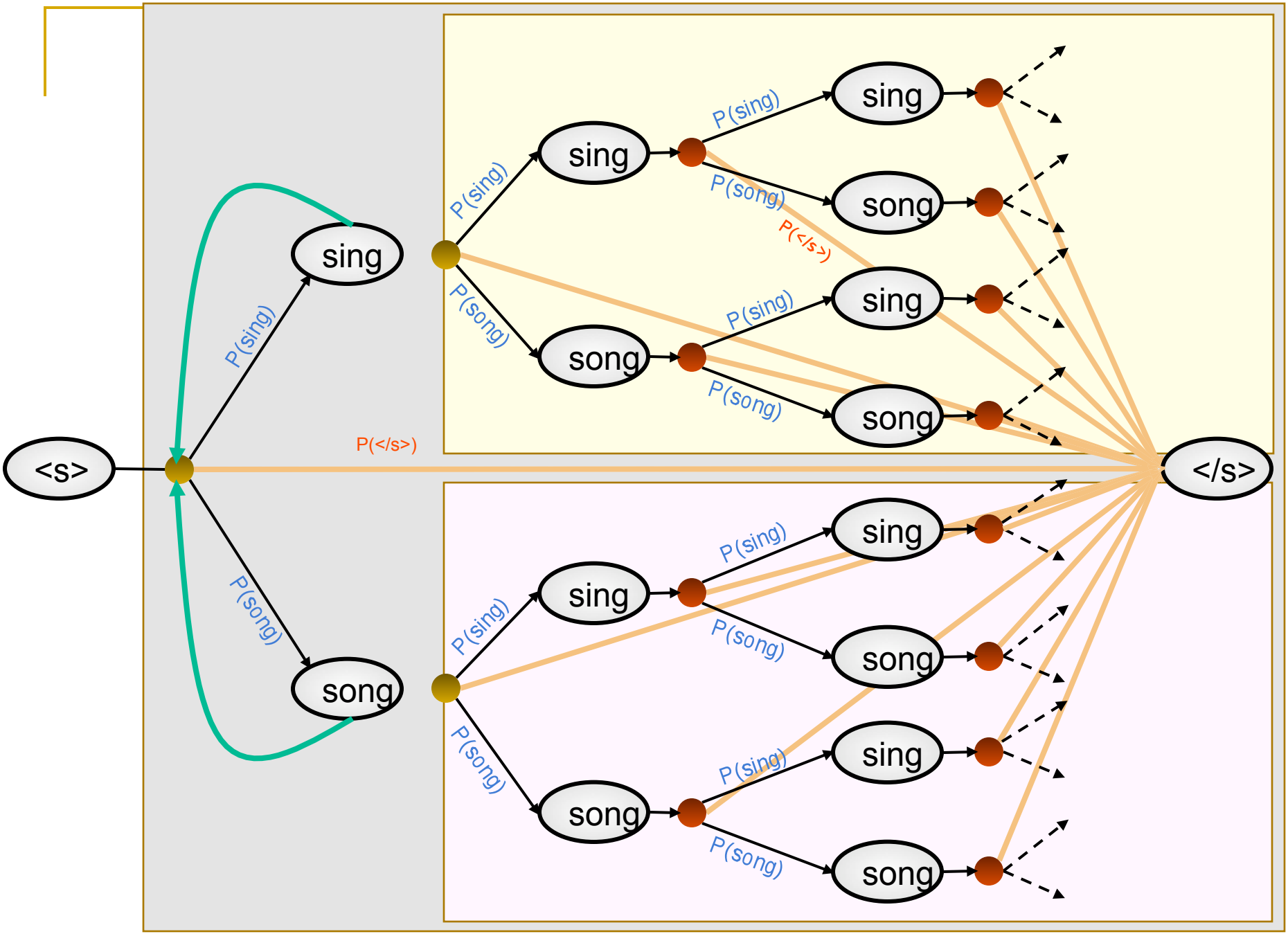
The two-word example as a full tree with a unigram LM

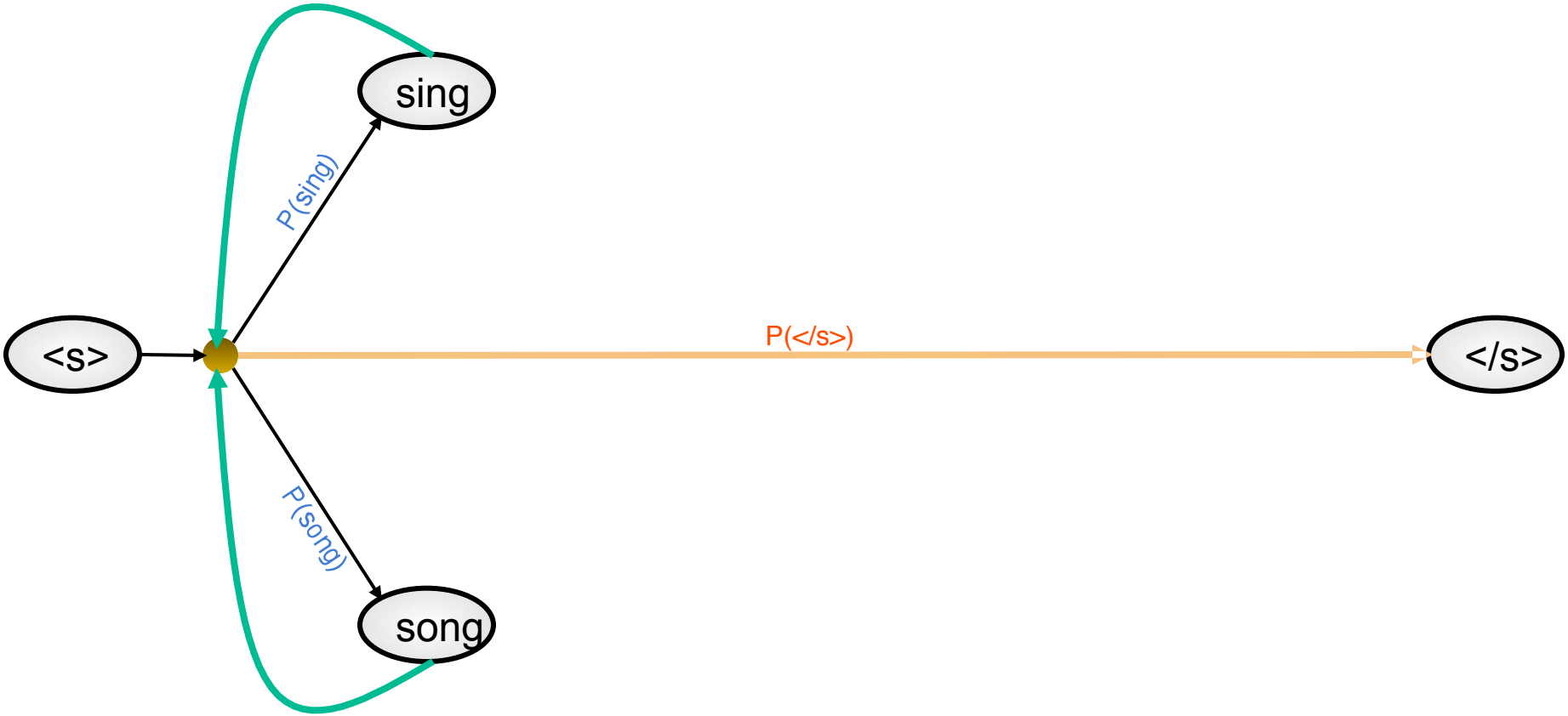


- The structure is recursive and can be collapsed

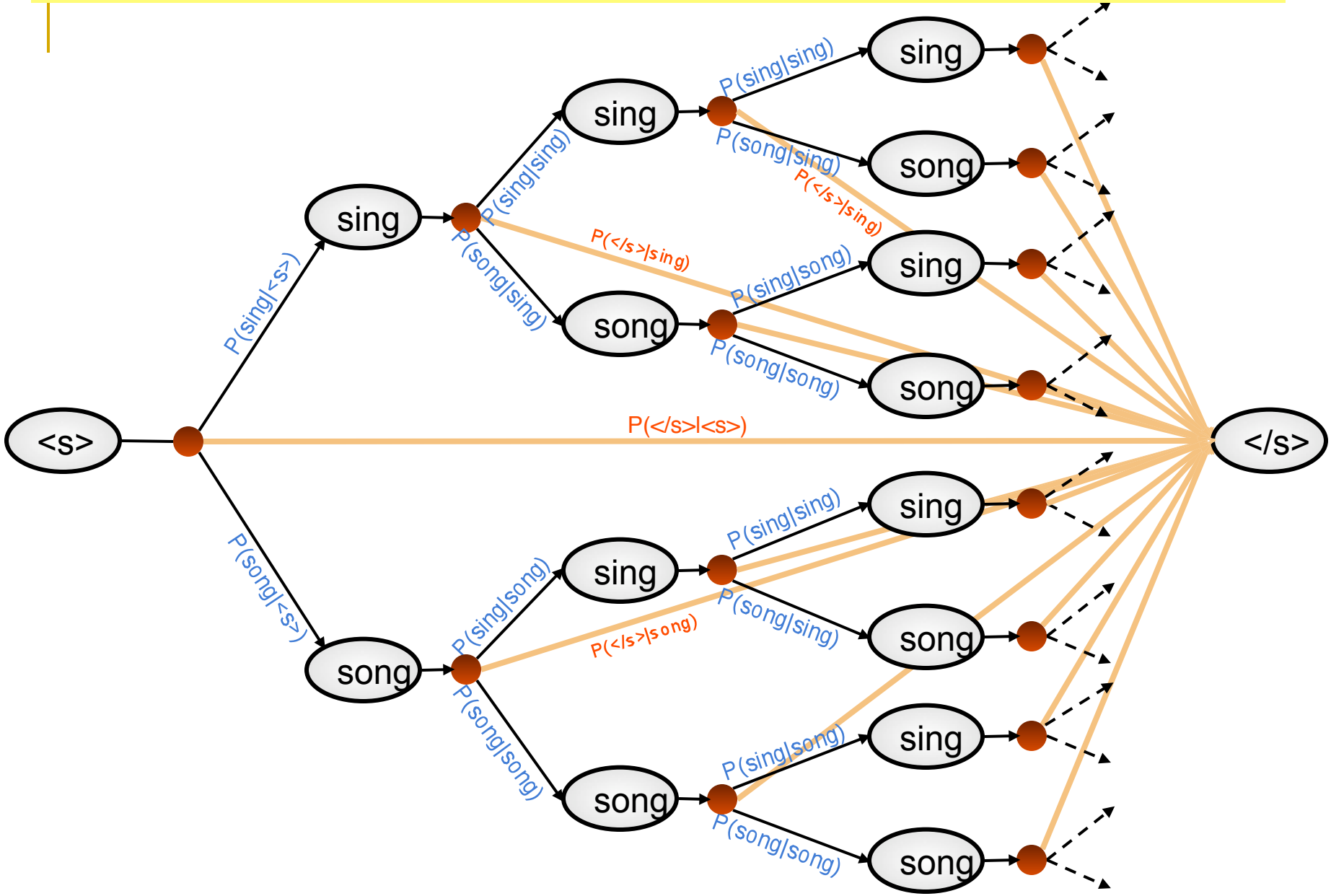

<s>
sing
song
P(sing)
P(song)
P(</s>)
</s>


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

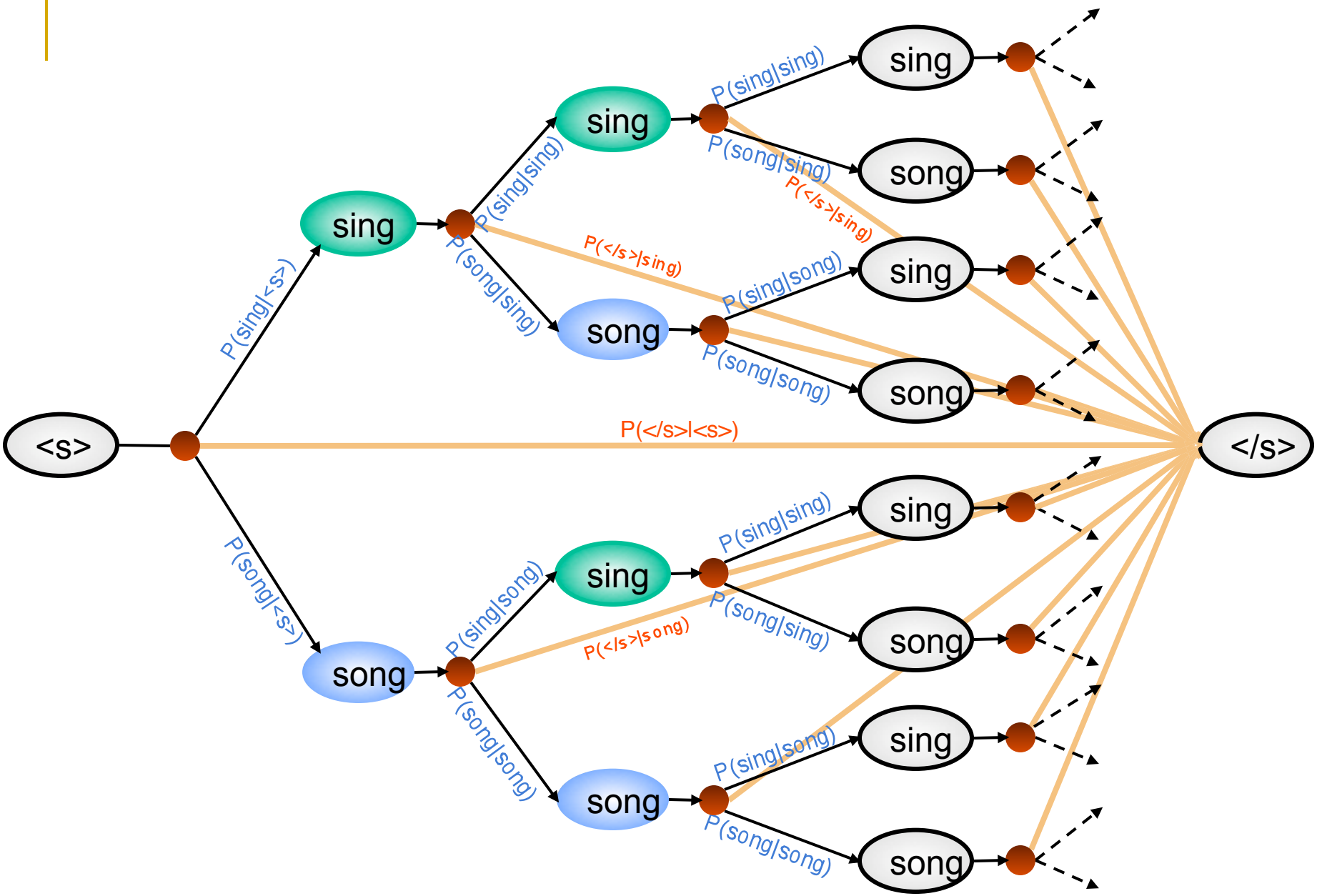

<s>
sing
song
P(sing)
P(song)
P(</s>)
sing
song
P(sing)
P(song)
P(</s>)
sing
song
P(sing)
P(song)
sing
song
P(sing)
P(song)
</s>

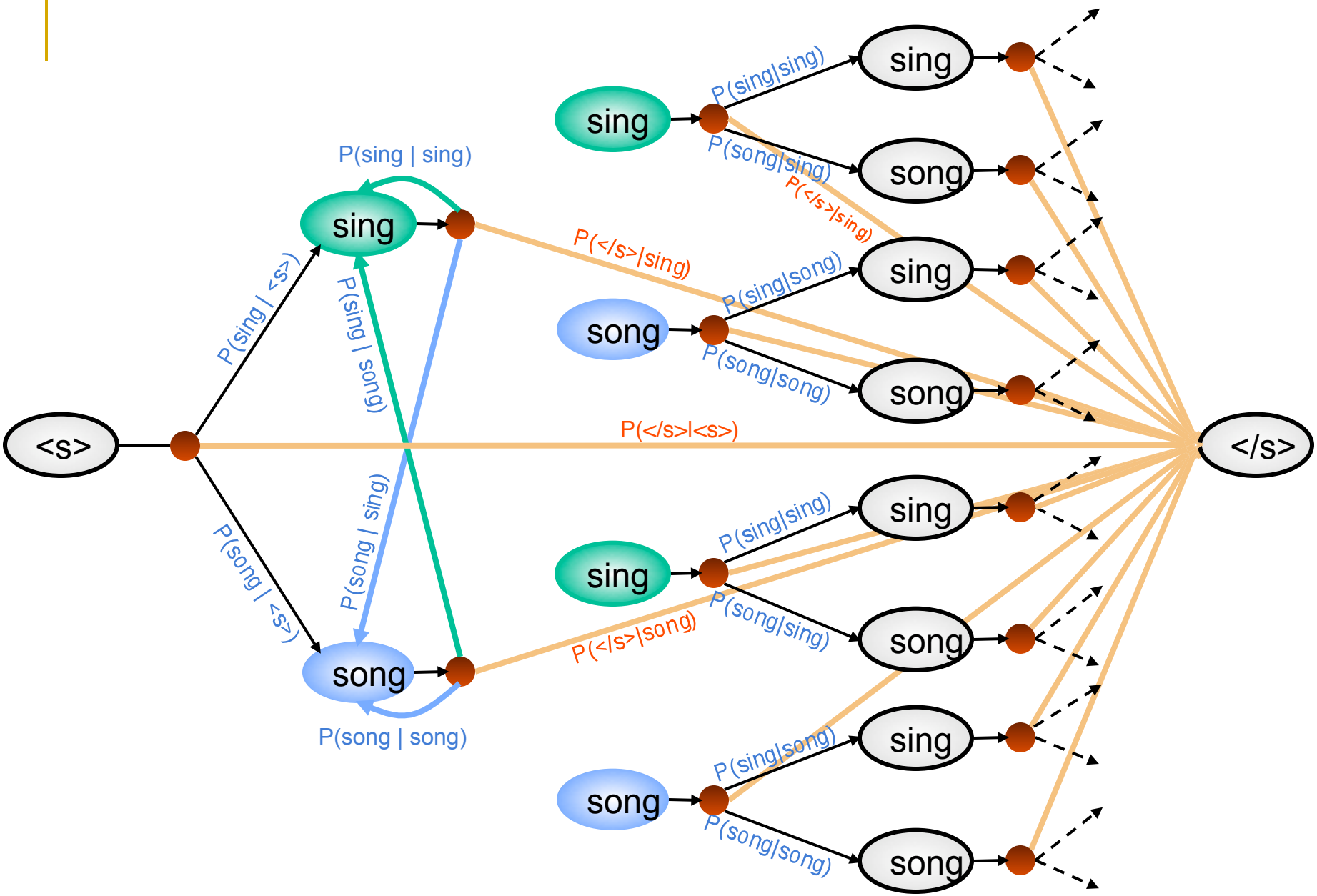

<s>
sing
song
P(sing)
P(song)
P(</s>)
</s>

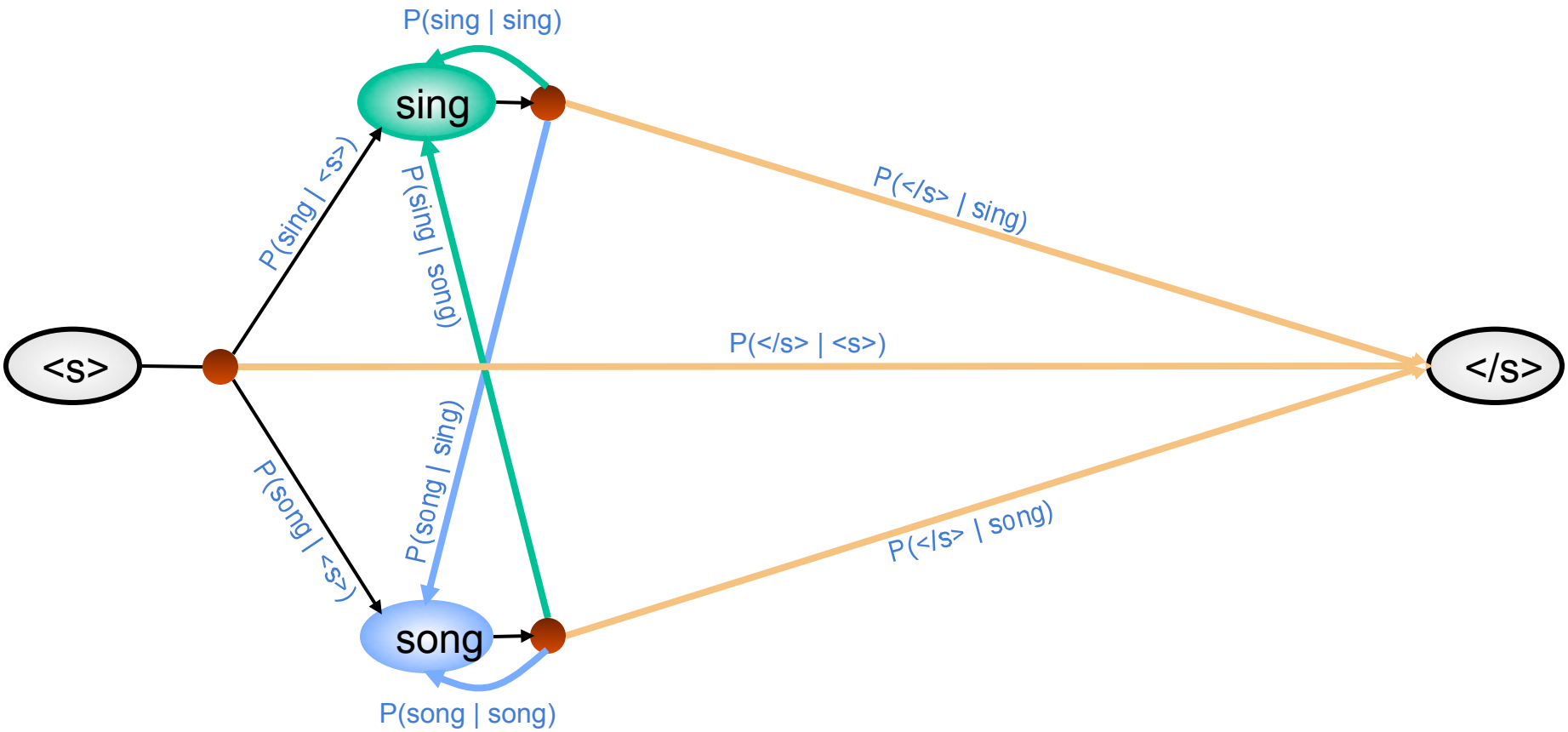
The two-word example as a full tree with a bigram LM



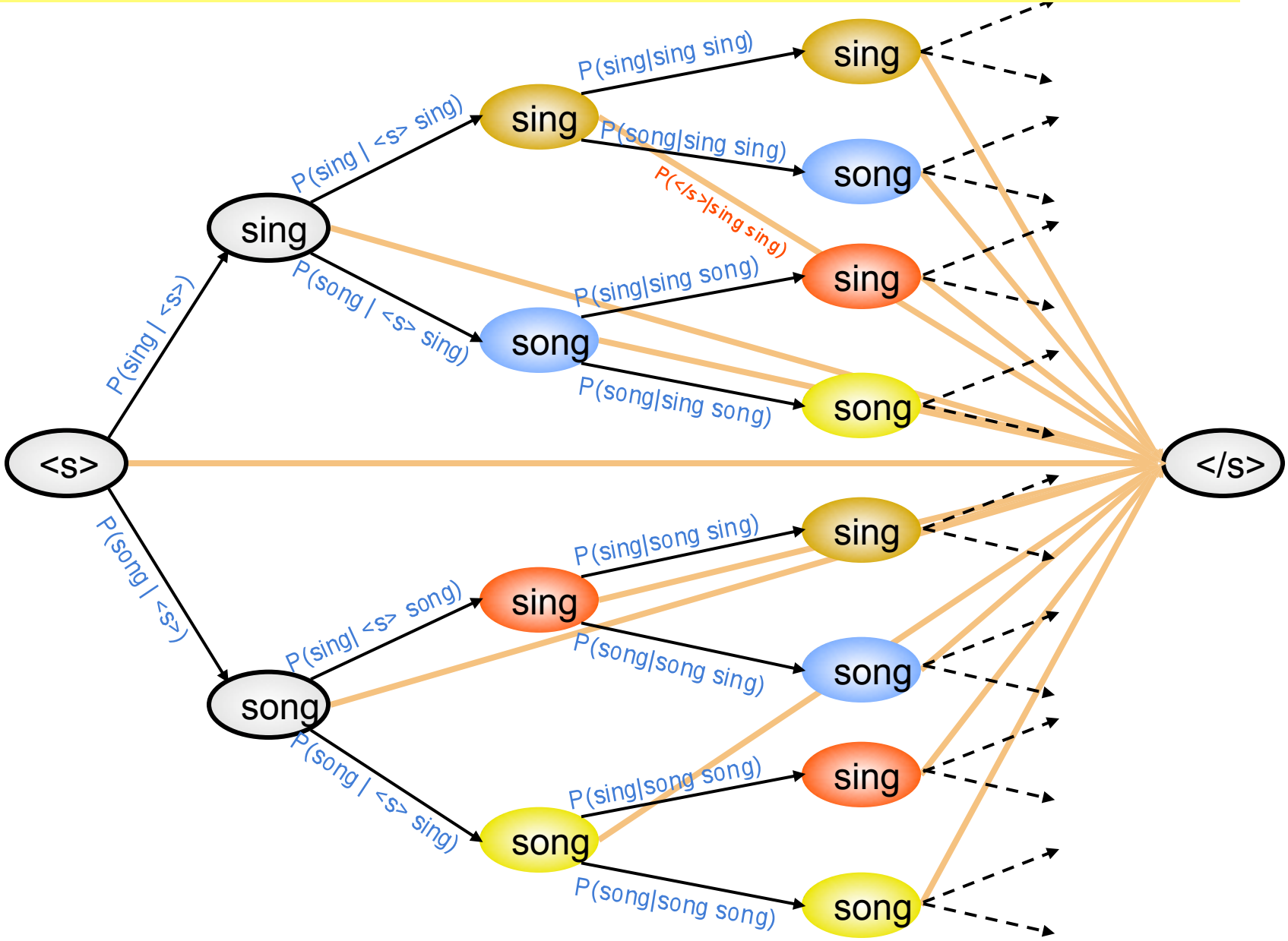
- The structure is recursive and can be collapsed


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

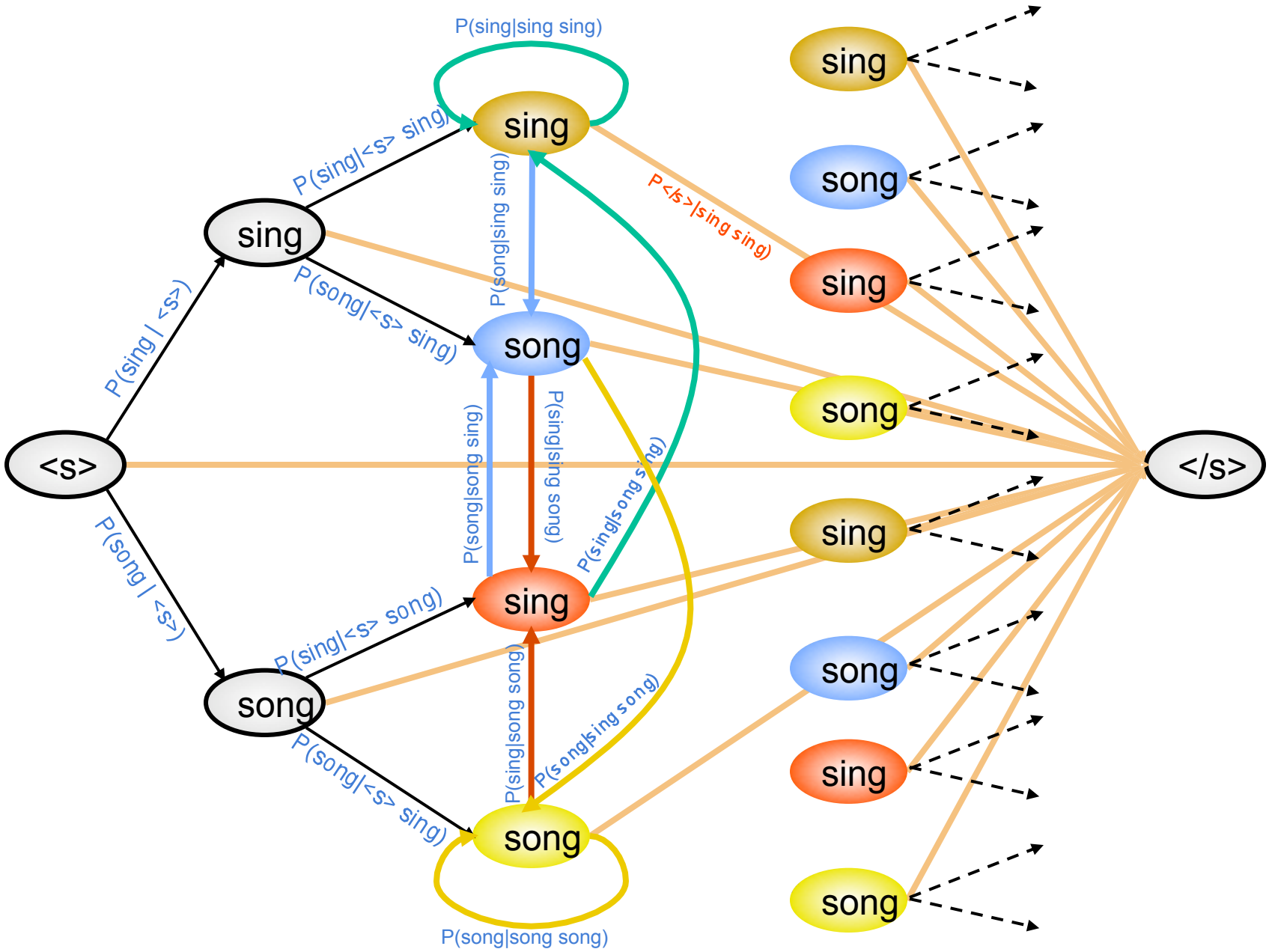

<s>
sing
song
P(sing | sing)
P(song | song)
P(sing | <s>)
P(song | <s>)
P(sing | sing)
P(song | sing)
P(song | song)
P(</s>|sing)
P(</s>|song)
P(</s>|<s>)
sing
song
P(sing|sing)
P(song|sing)
P(sing|song)
P(song|song)
P(</s>|sing)
P(</s>|song)
</s>

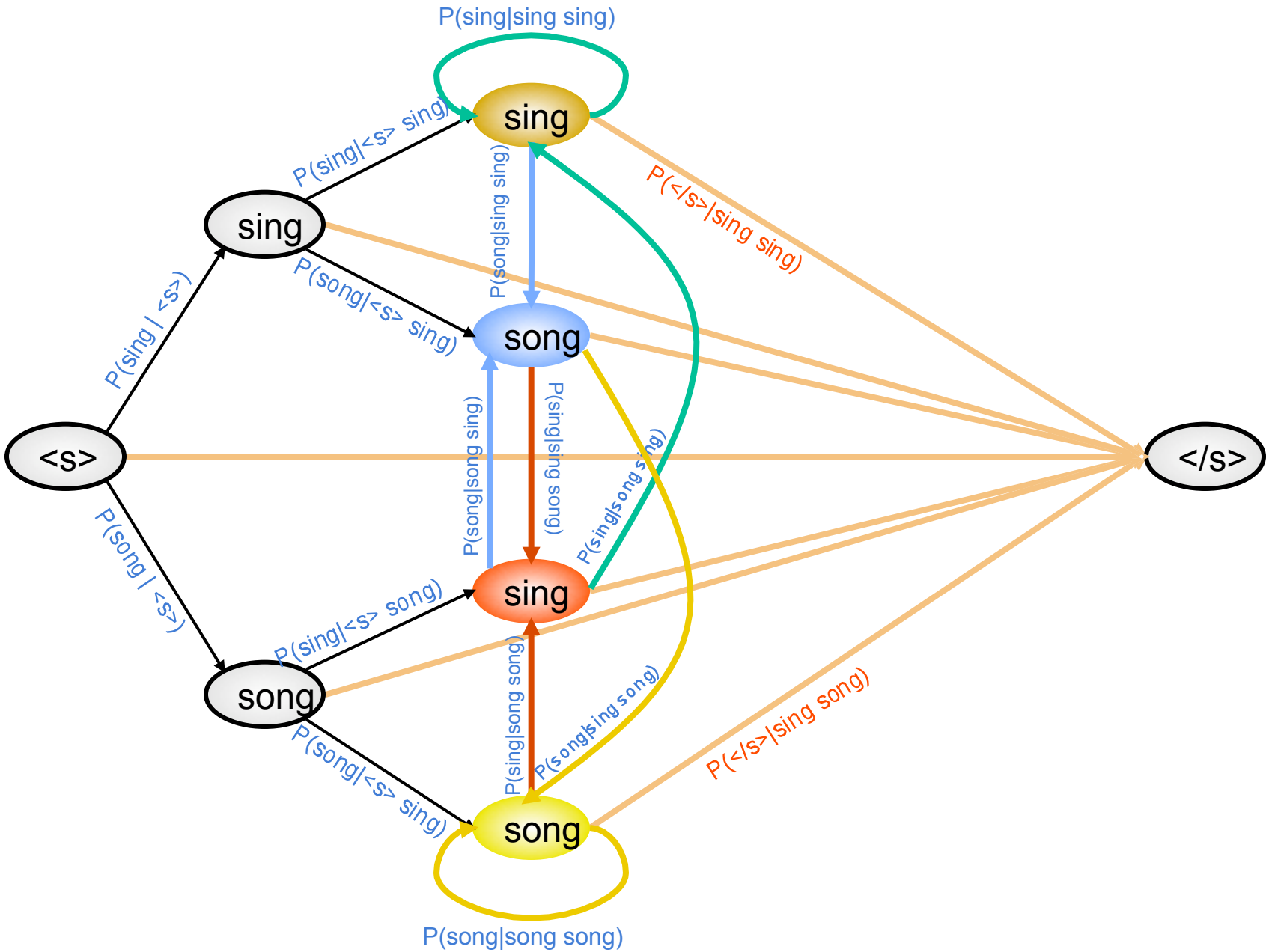

P(sing | sing)
sing
P(sing | <s>)
P(sing | song)
P(</s> | sing)
<s>
P(</s> | <s>)
</s>
P(song | <s>)
P(song | sing)
P(</s> | song)
song
P(song | song)

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



- The structure is recursive and can be collapsed


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


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

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

- 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

To Build a Speech Recognizer

- Train word HMMs from many training instances
  - Typically one trains HMMs for individual phonemes, then concatenates them to make HMMs for words
  - Recognition, however is almost always done with WORD HMMs (and not phonemes as is often misunderstood)
- Train or decide a language model for the task
  - Either a simple grammar or an N-gram model
- Represent the language model as a compact graph
- Introduce the appropriate HMM for each word in the graph to build a giant HMM

- Use the Viterbi algorithm to find the best state sequence (and thereby the best word sequence) through the graph!