Speech Recognition for Dummies

Bhiksha Raj

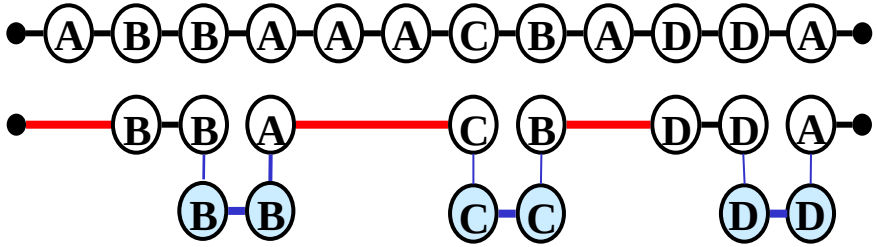
String Matching

- 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

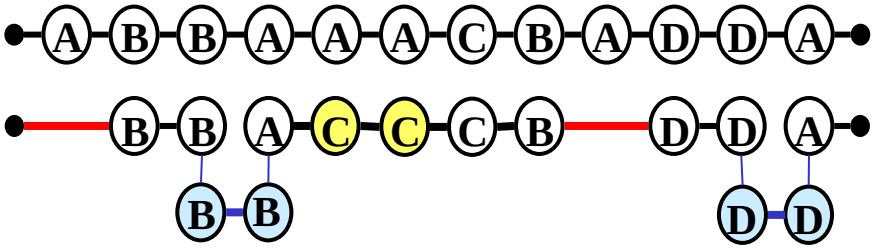
- Cost includes the costs of “insertion”, “Deletion”, “Substitution” and “Match”

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: $2I(C)+ 2I(B) + 2I(D) + 4D(A) + 3M(B) + M(A) + M(C) + 2M(D)$



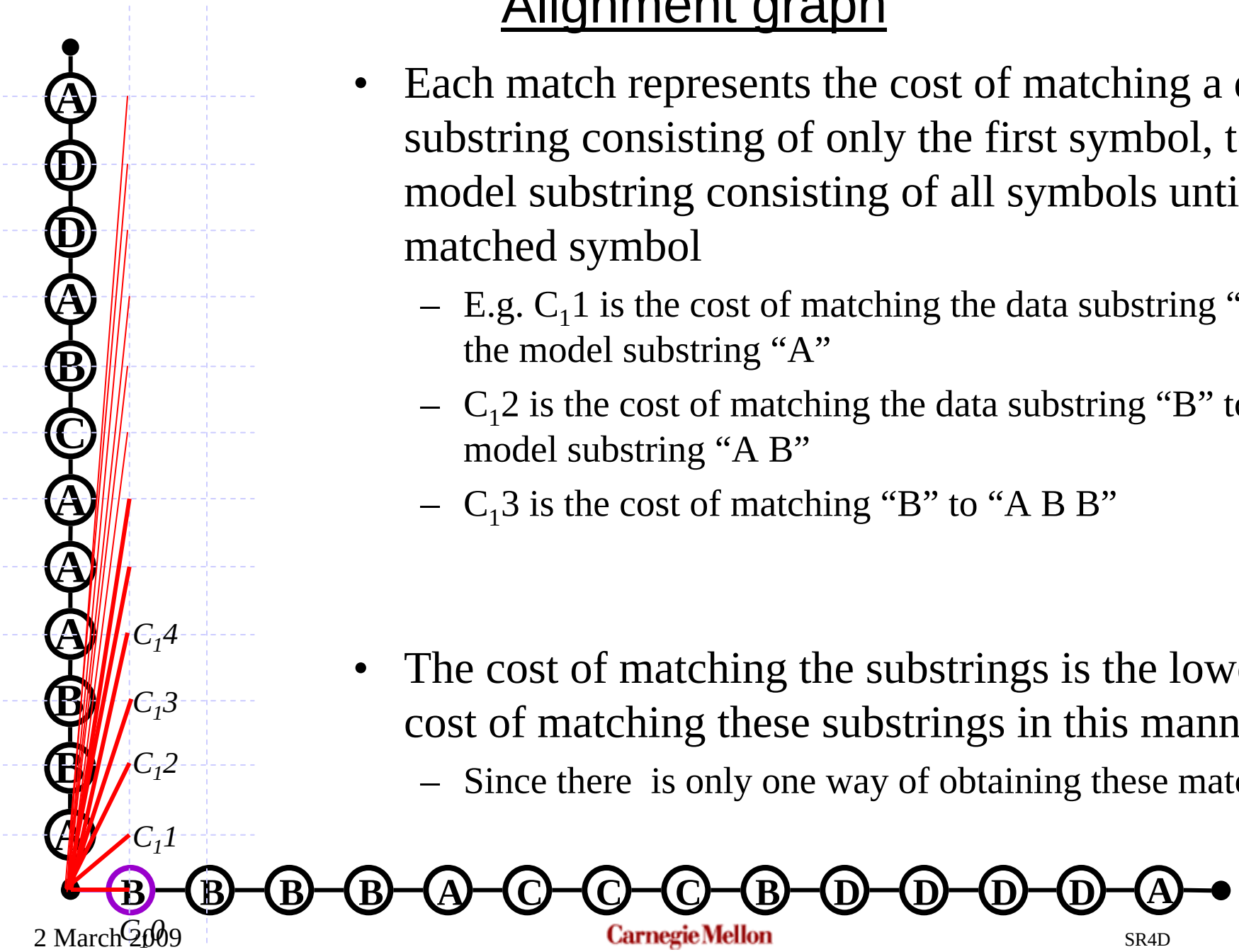
- Match 2:
  - Insertions: B, B, D, D
  - Deletions: A, A
  - Substitutions: (A,C), (A,C)
  - Matches: B, B, A, C, B, D, D, A
  - Total cost: $2I(B)+ 2I(D) + 2D(A) + 2S(A,C) + 3M(B) + 2M(A) + M(C) + 2M(D)$



Computing the minimum cost

- The cost of matching a data string to a model string is the minimum distortion that must be imposed on the model string to convert it to the data string
- How does one compute the minimum distortion?
  - Exponentially large number of possibilities for matching two strings
  - Exhaustive evaluation of the cost of all possibilities to identify the minimum cost match (and the corresponding cost) 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

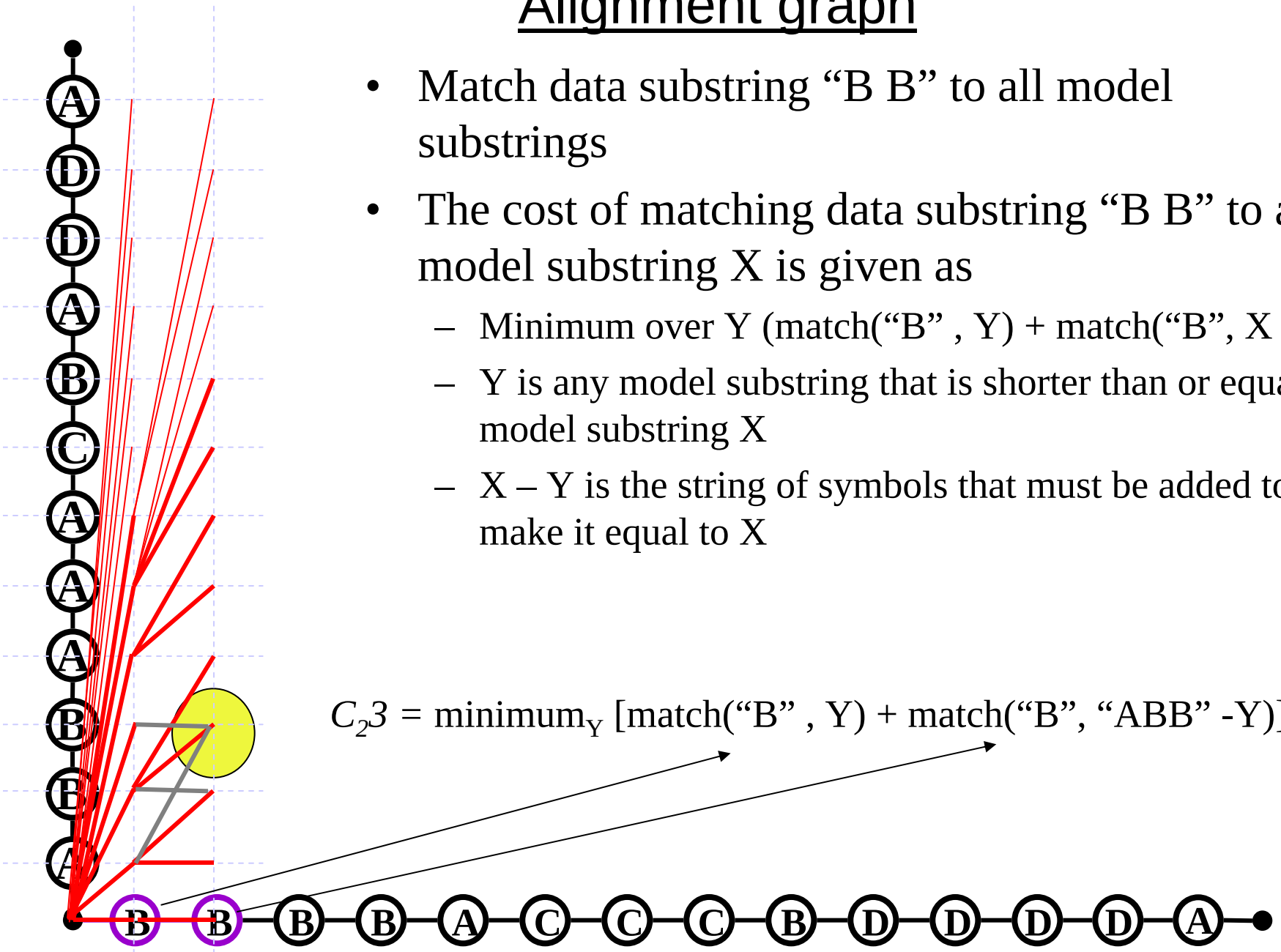
Alignment graph



- 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

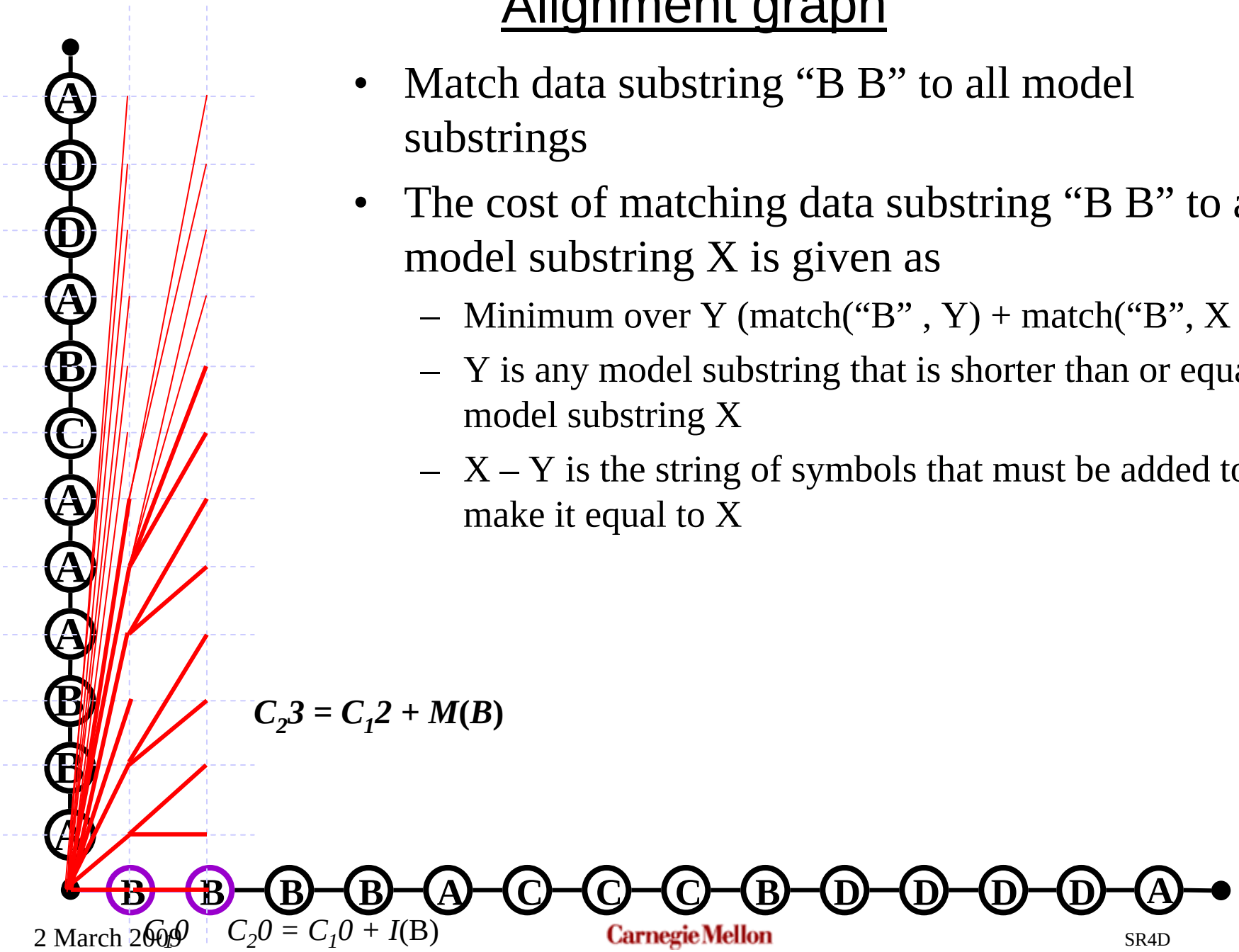
Alignment graph

- 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



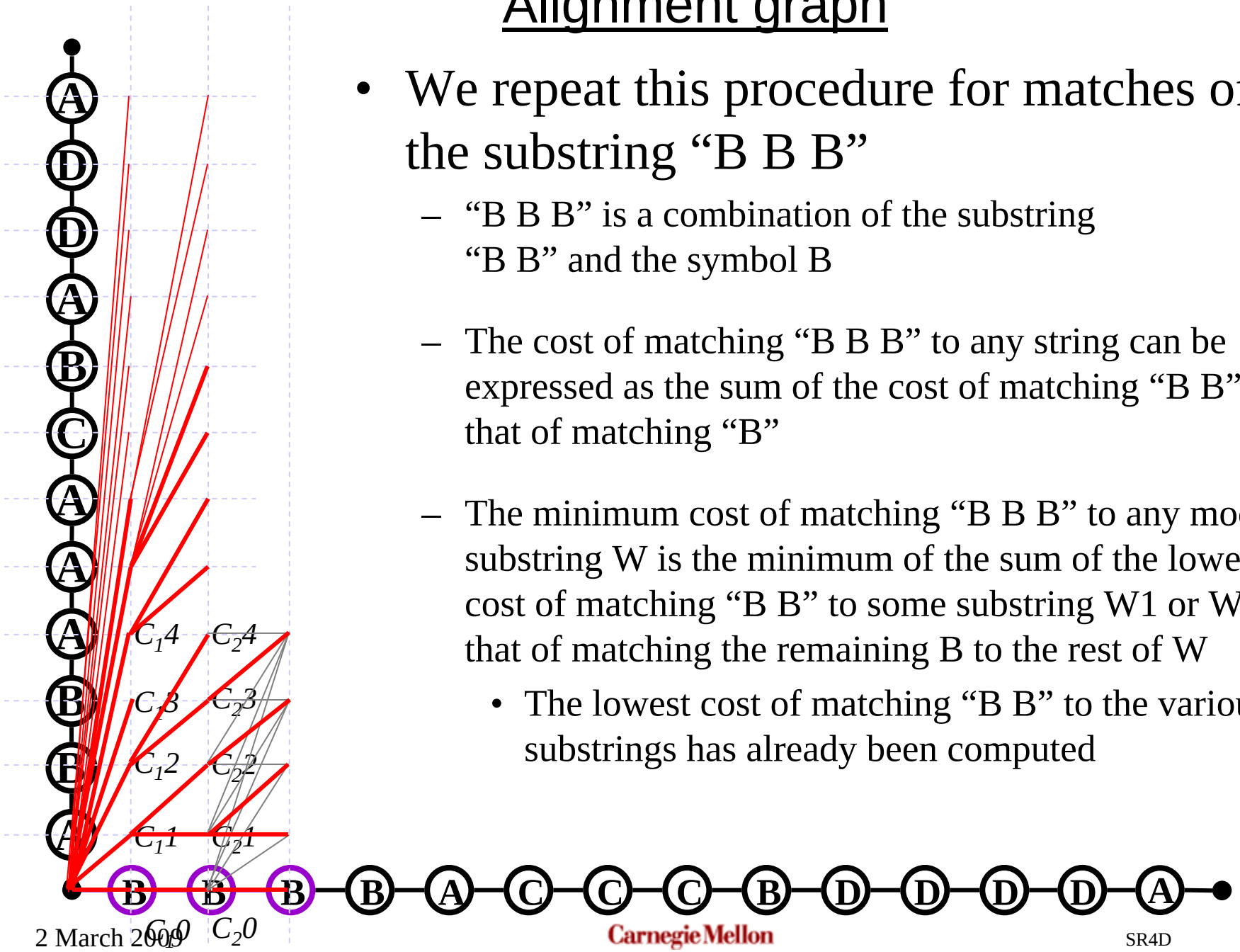
$C_2 3$ = minimum$_Y$ [match(“B” , Y) + match(“B”, “ABB” -Y)]

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

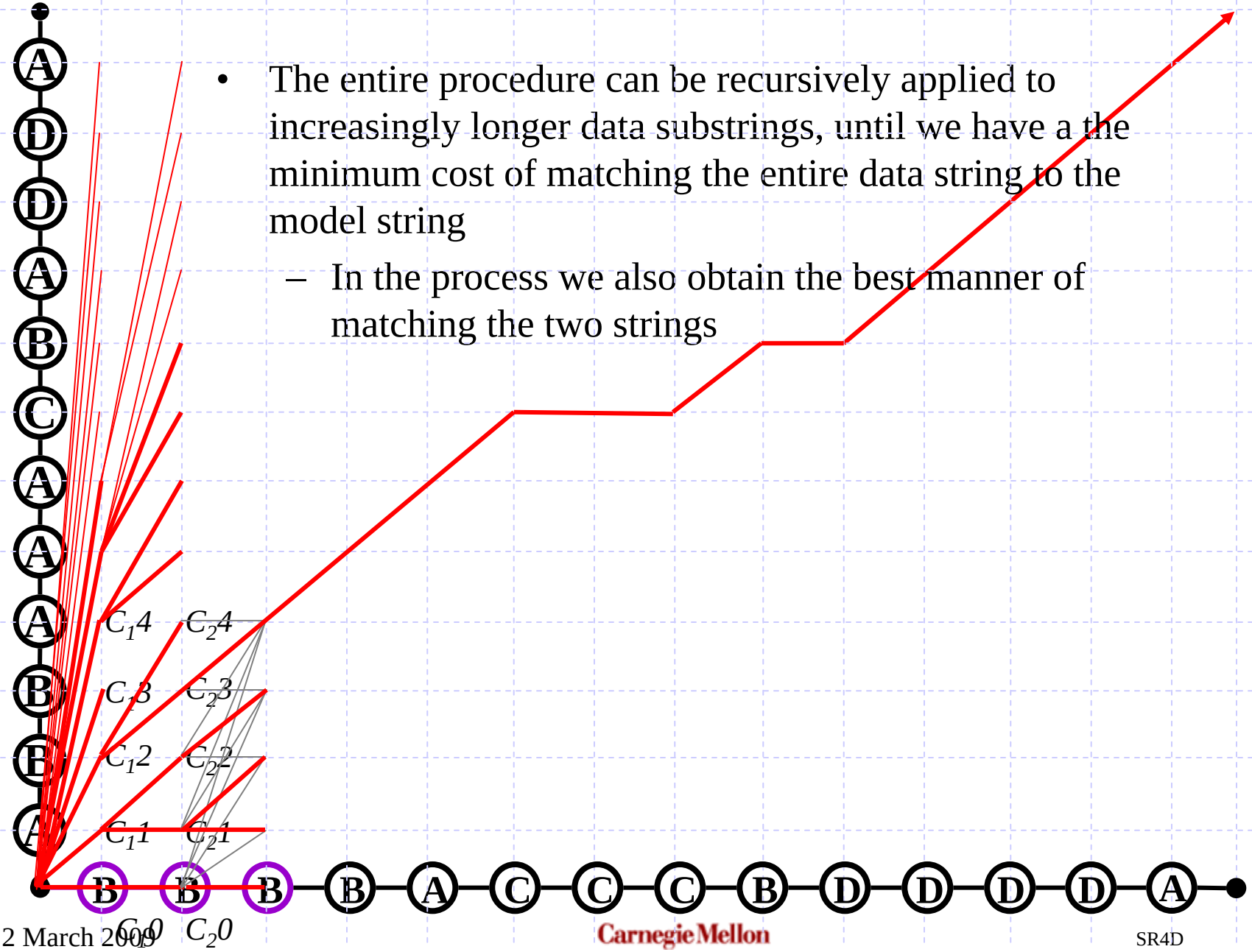
Alignment graph



- 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 can be expressed as the sum of the cost of matching “B B” and that of matching “B”
  - The minimum cost of matching “B B B” to any model substring W is the minimum of the sum of the lowest cost of matching “B B” to some substring W1 or W, and that 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

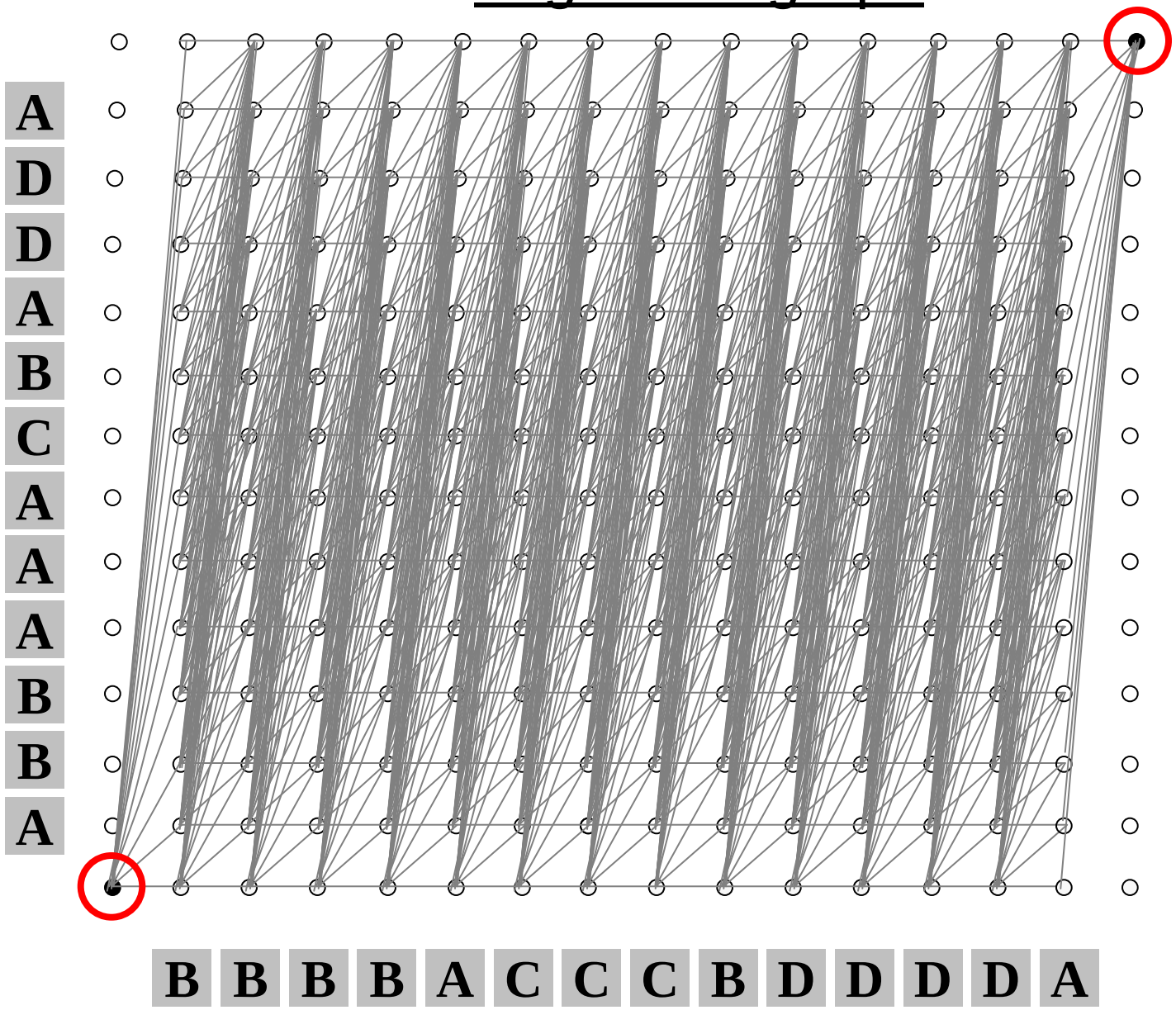


Aligning two strings

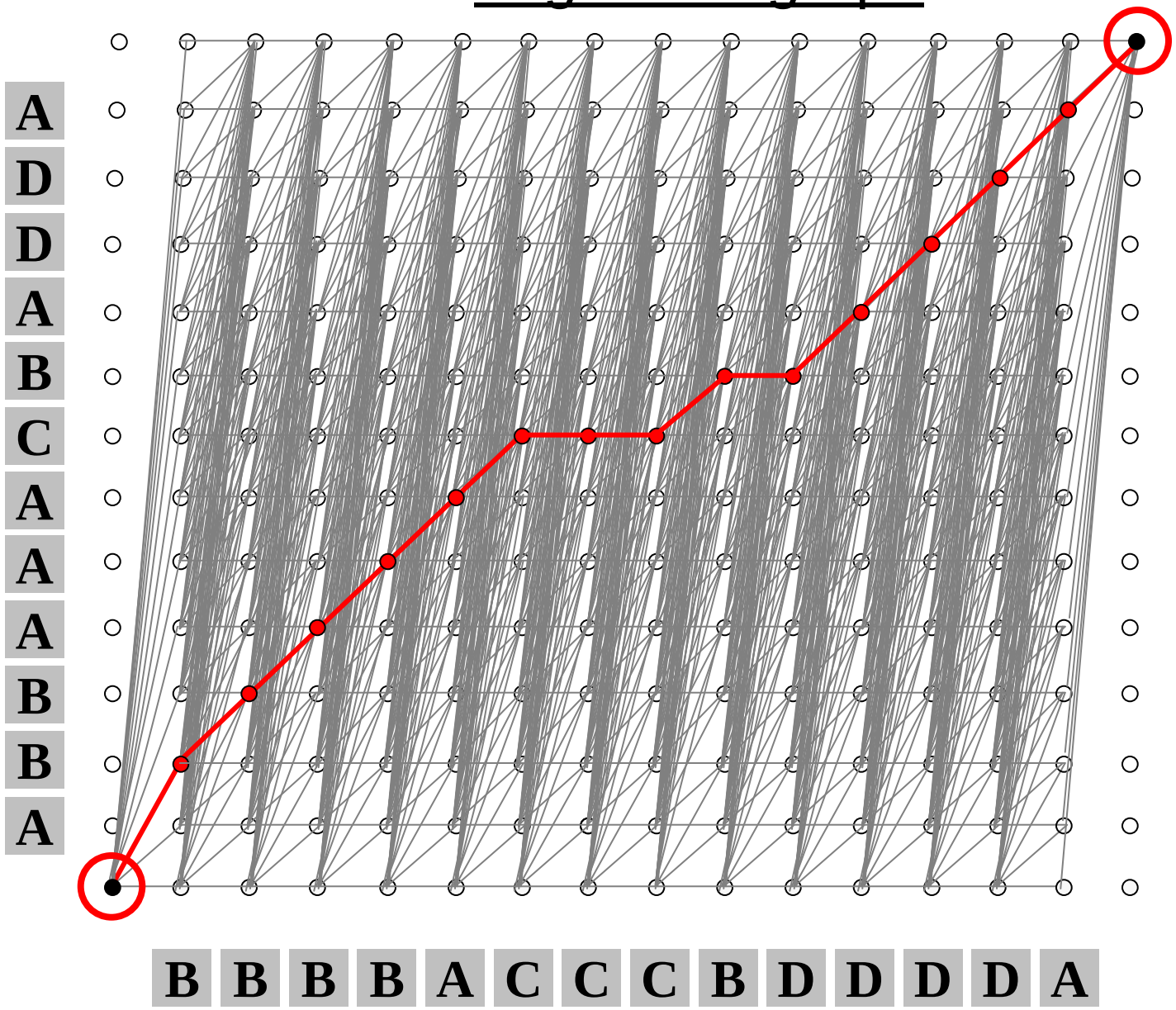
- The alignment process can be viewed as graph search

- N.B: Only in this formulation
  - Conventional string matching uses the “Edit Distance”
  - String matching with the edit distance is not strictly representable as search over a static graph
    - Node score depends on the specific incoming edge

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) is not necessarily equal to Distance(string2, string1)
- The graph may be constructed in other ways
  - Symmetric graphs, where symbols on the horizontal axis may also be skipped
- Additional constraints may be incorporated into the graph
  - 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: $(\Sigma_i w_i|A_i - B_i|^n)^{1/n}$

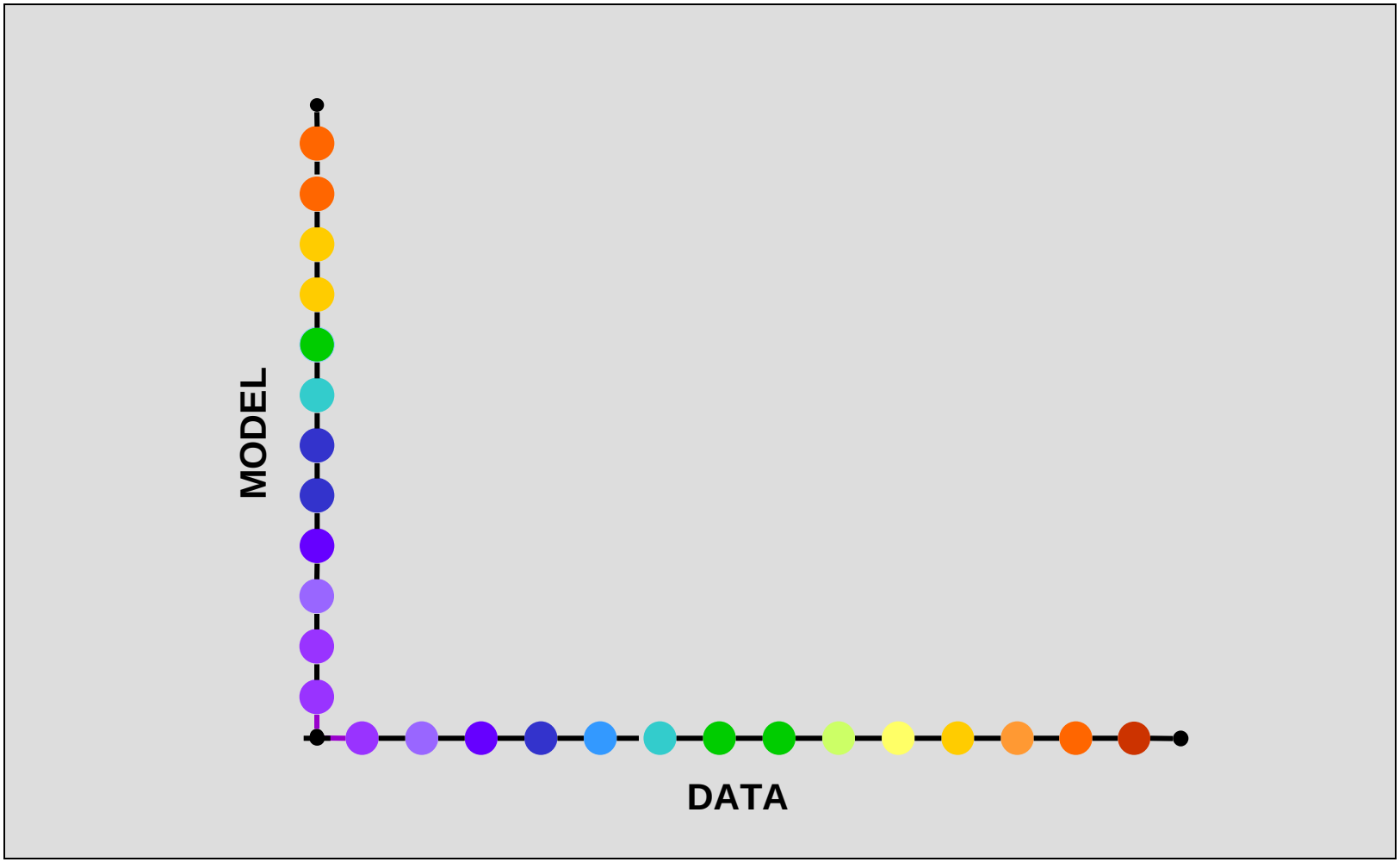
DTW and speech recognition

- 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 cepstrals 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

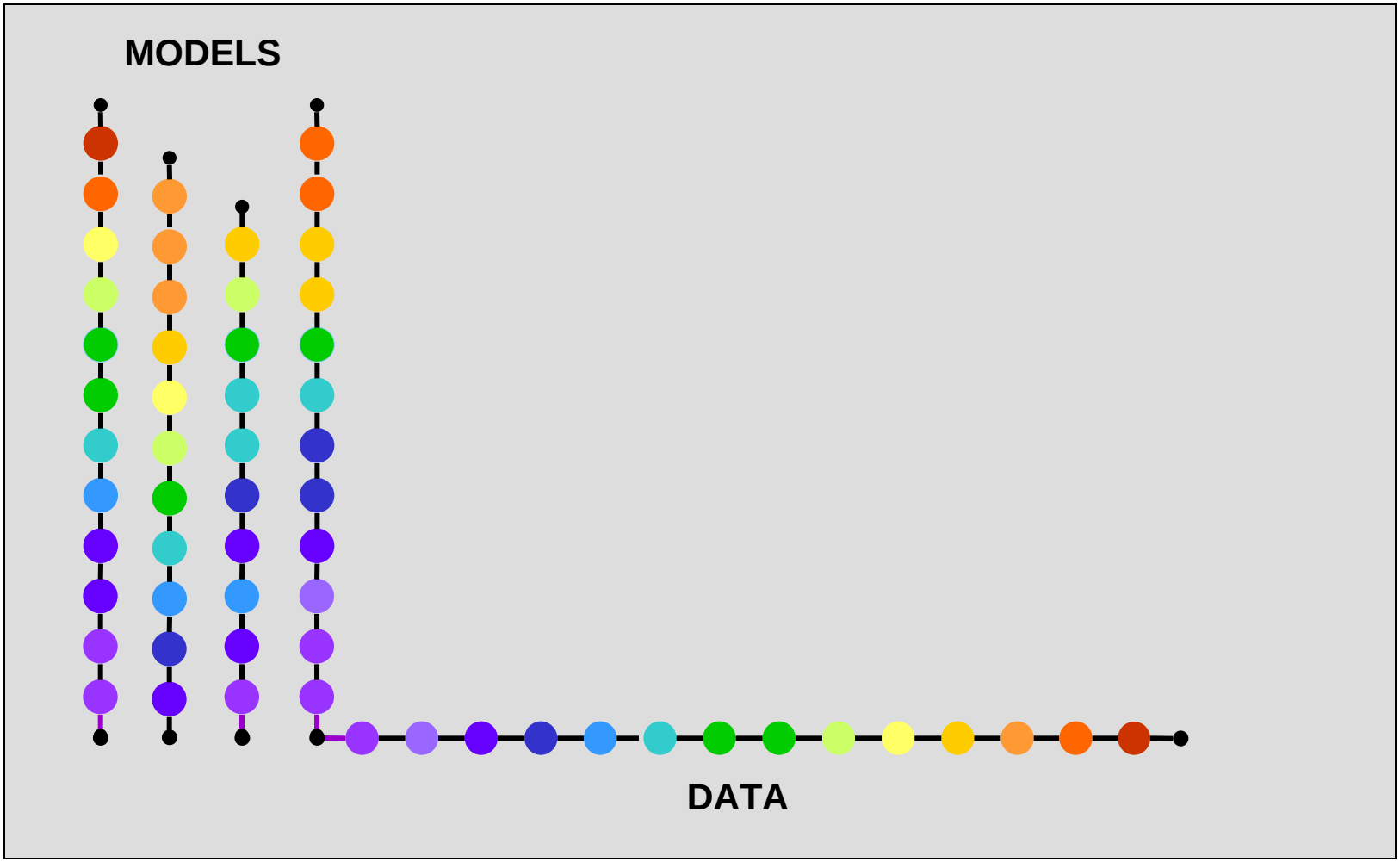


The template (model) is matched against the data string to be recognized
Select the template with the lowest cost of match

Using Multiple Templates

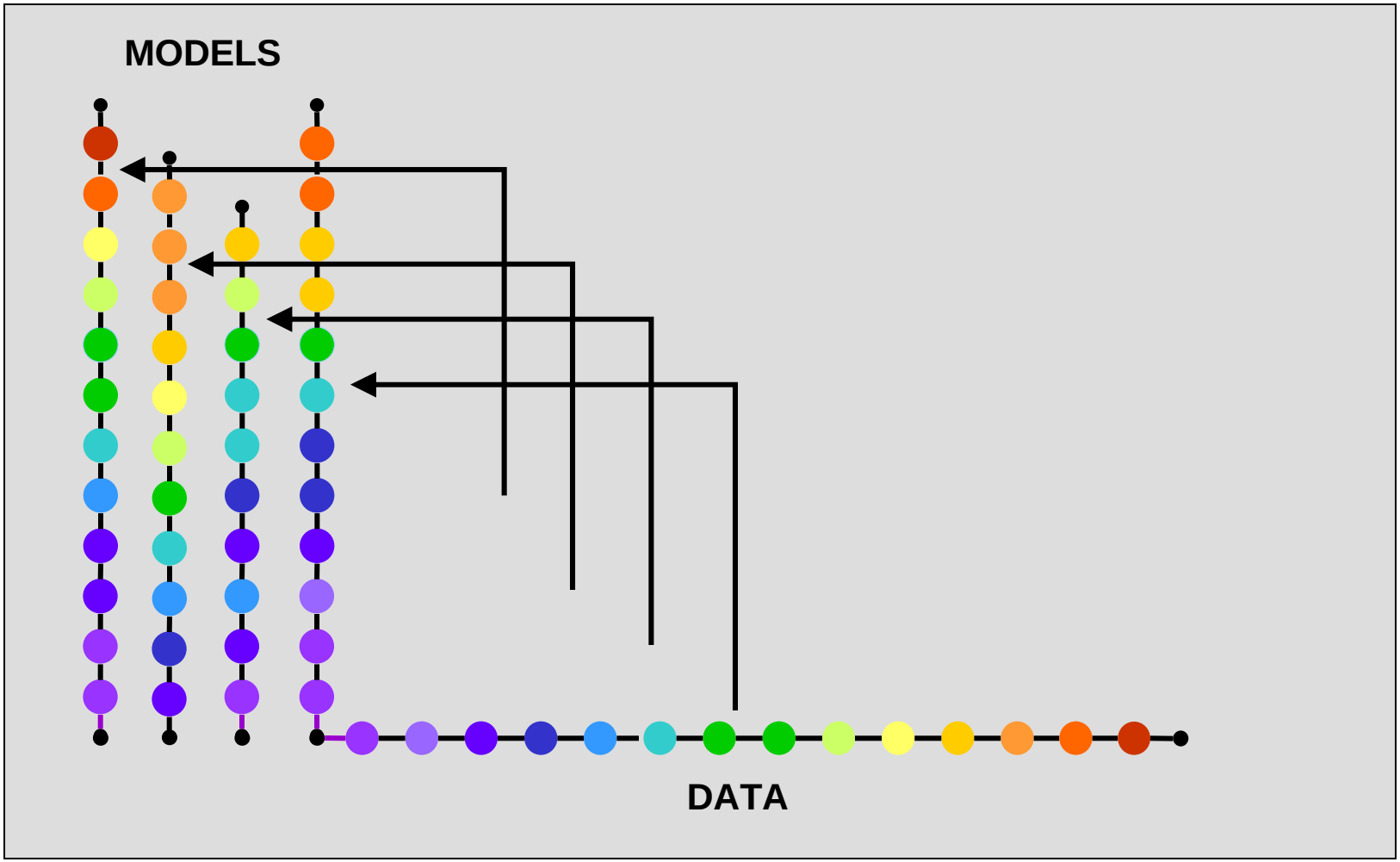
- A person may utter a word (e.g. ZERO) in multiple ways
  - In fact, one never utters the word twice in exactly the same way
- Store *multiple* templates for each word
  - Record 5 instances of “ZERO”, five of “ONE” etc.
- Recognition: For each word, select the template that is closest to the test utterance. Finally, select the word for which the distance from the closest template was minimum

DTW with multiple models



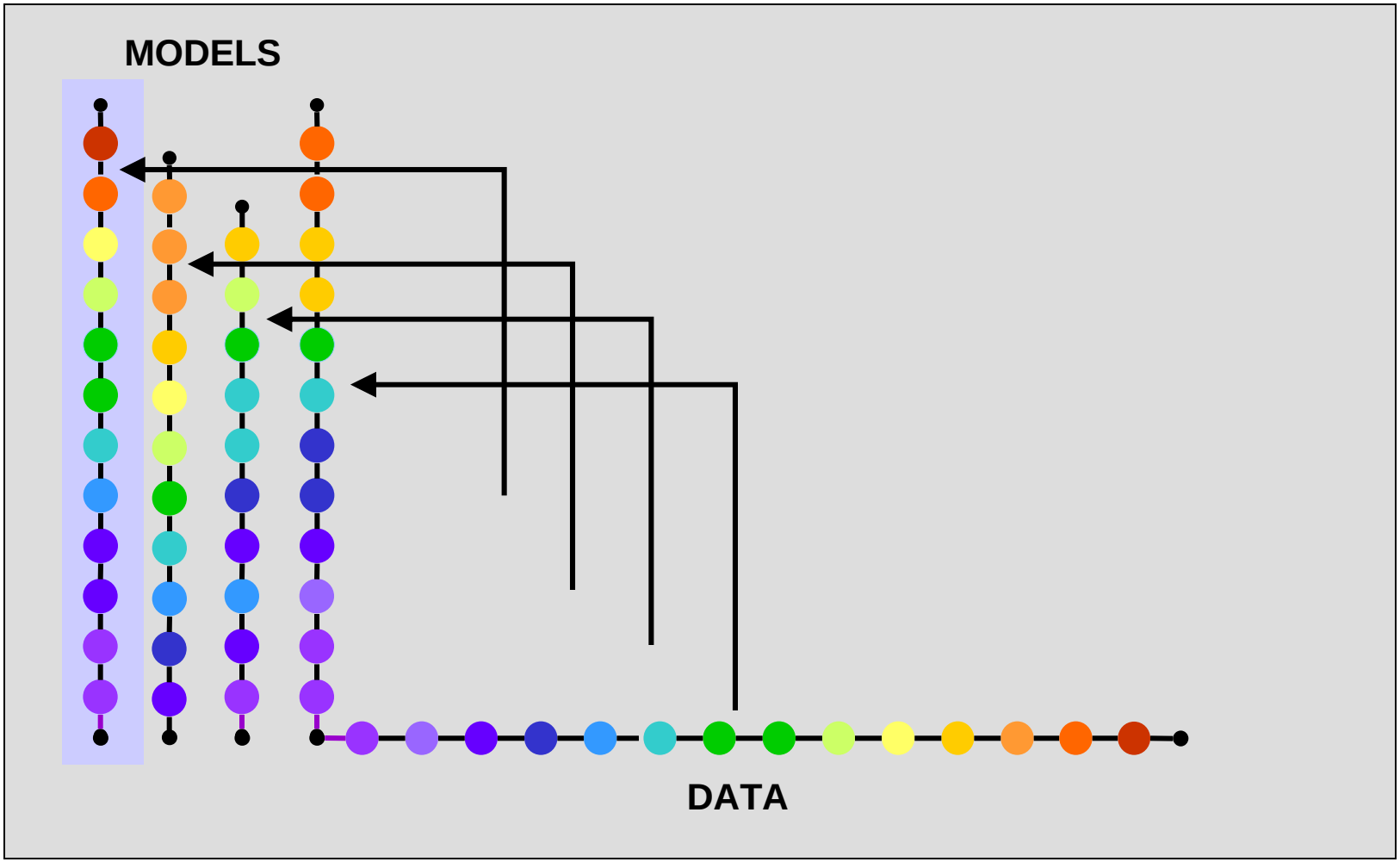
Evaluate all templates for a word against the data

DTW with multiple models



Evaluate all templates for a word against the data

DTW with multiple models

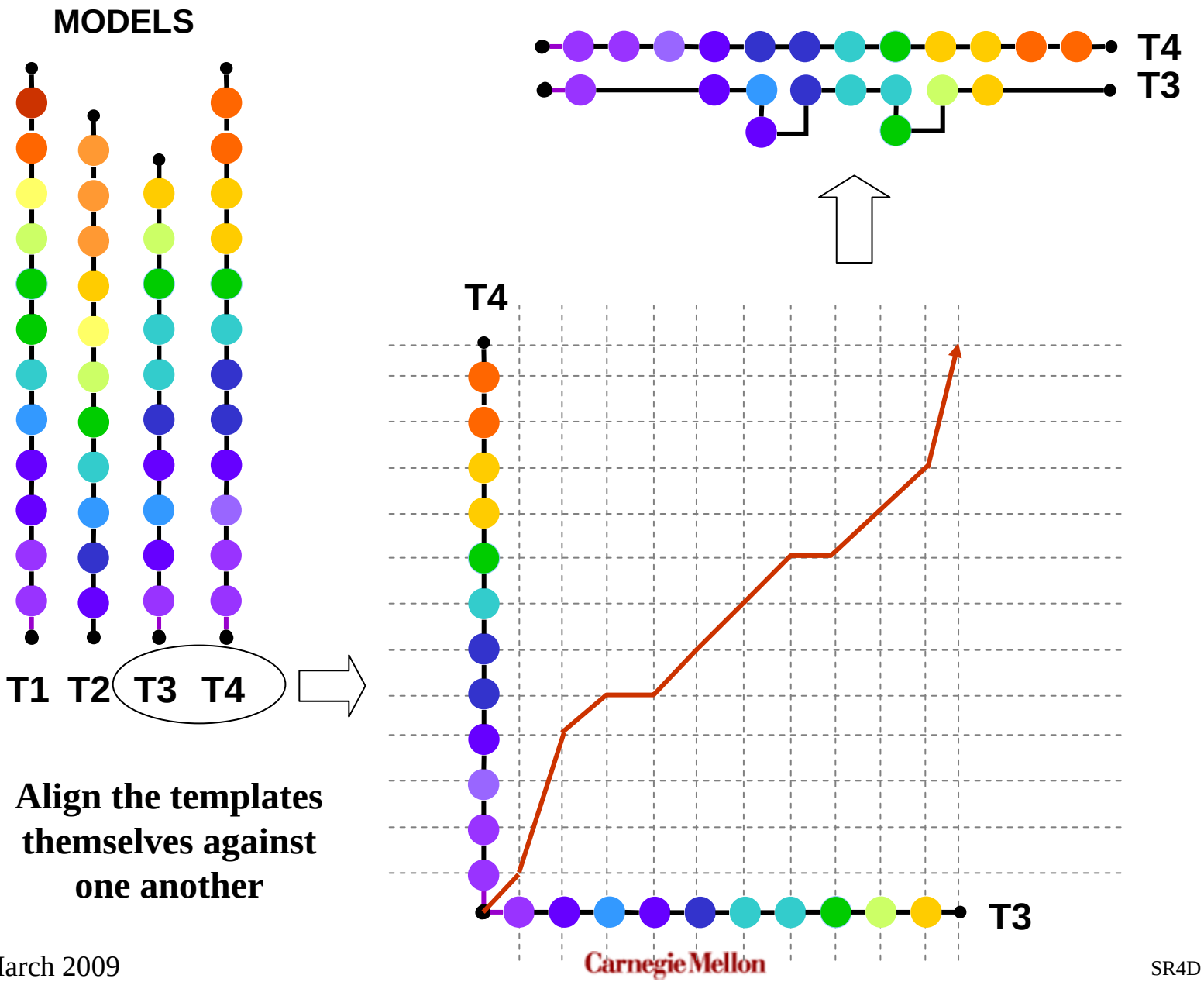


Evaluate all templates for a word against the data
Select the best fitting template. The corresponding cost is the cost of the match

The Problem with Multiple Templates

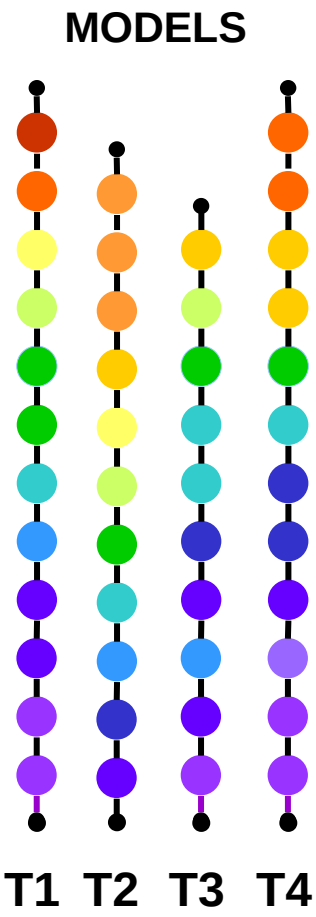
- Finding the closest template to a test utterance requires evaluation of all test templates
  - This is expensive
- Additionally, the set of templates may not cover all possible variants of the words
  - We would like some mechanism whereby we can generalize from the templates to something that represents even variants that have not hitherto been seen
- We do this by averaging the templates

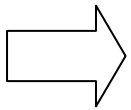
DTW with multiple models

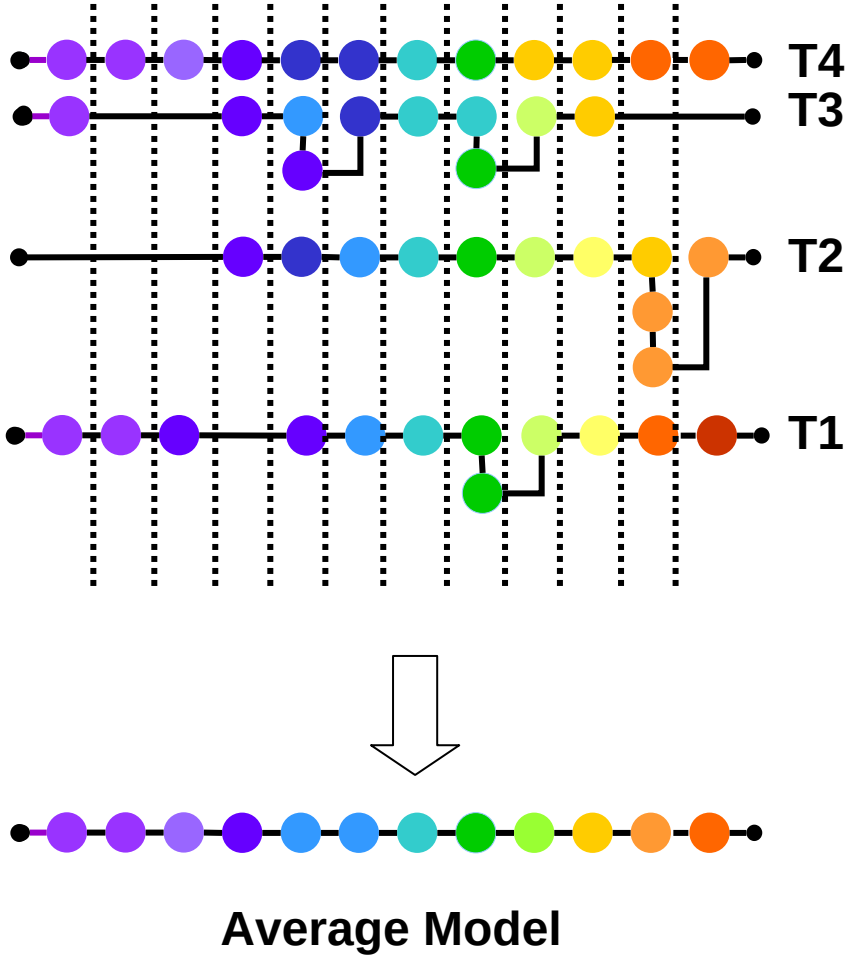


Align the templates themselves against one another

DTW with multiple models



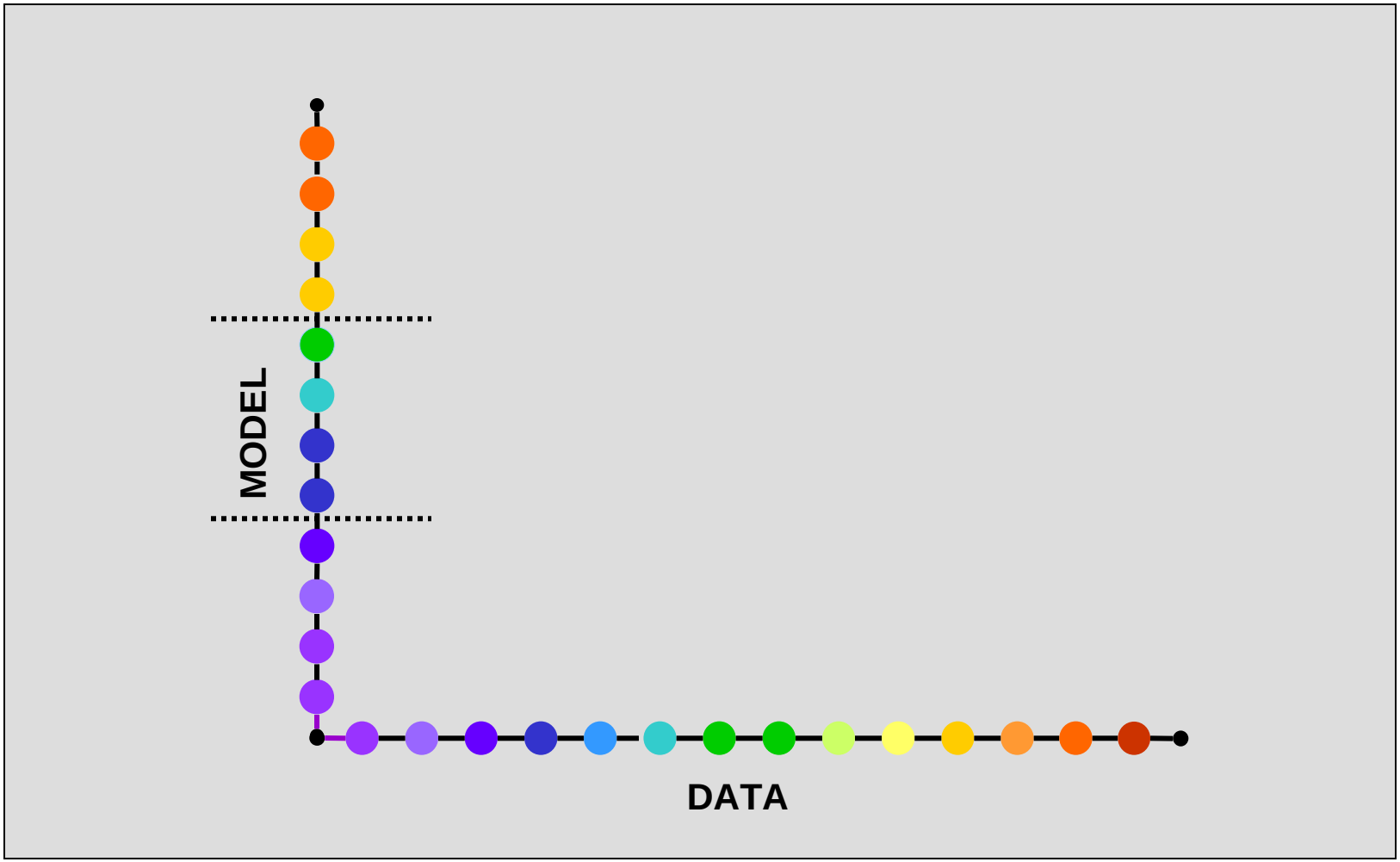




Align the templates themselves against one another

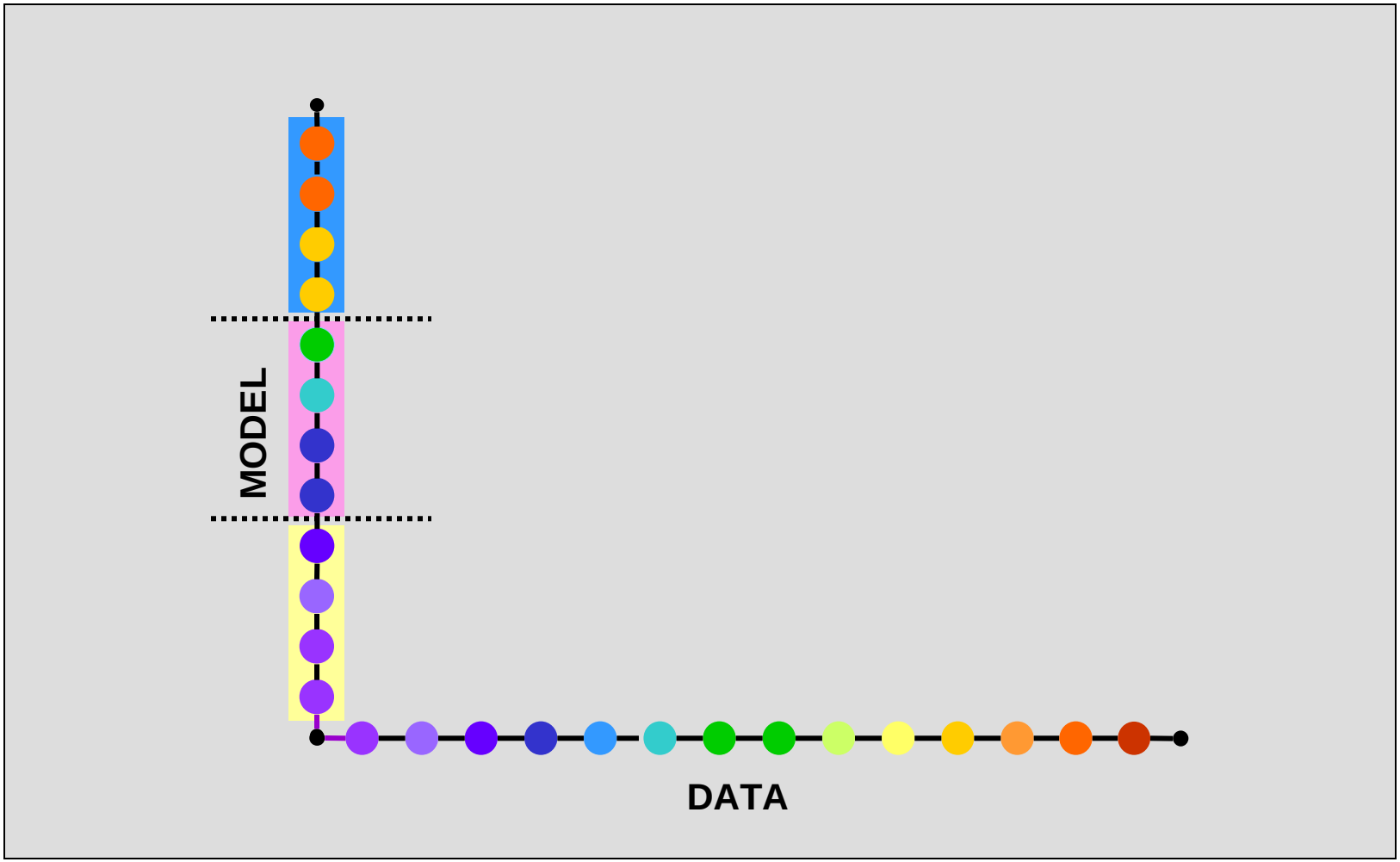
Average the aligned templates

DTW with one model



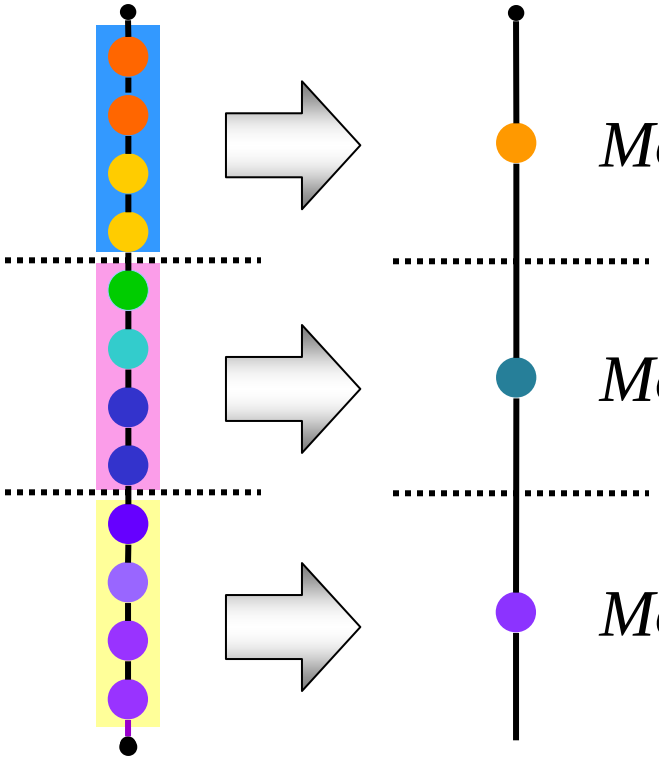
A SIMPLER METHOD: Segment the templates themselves and average within segments

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



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

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

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

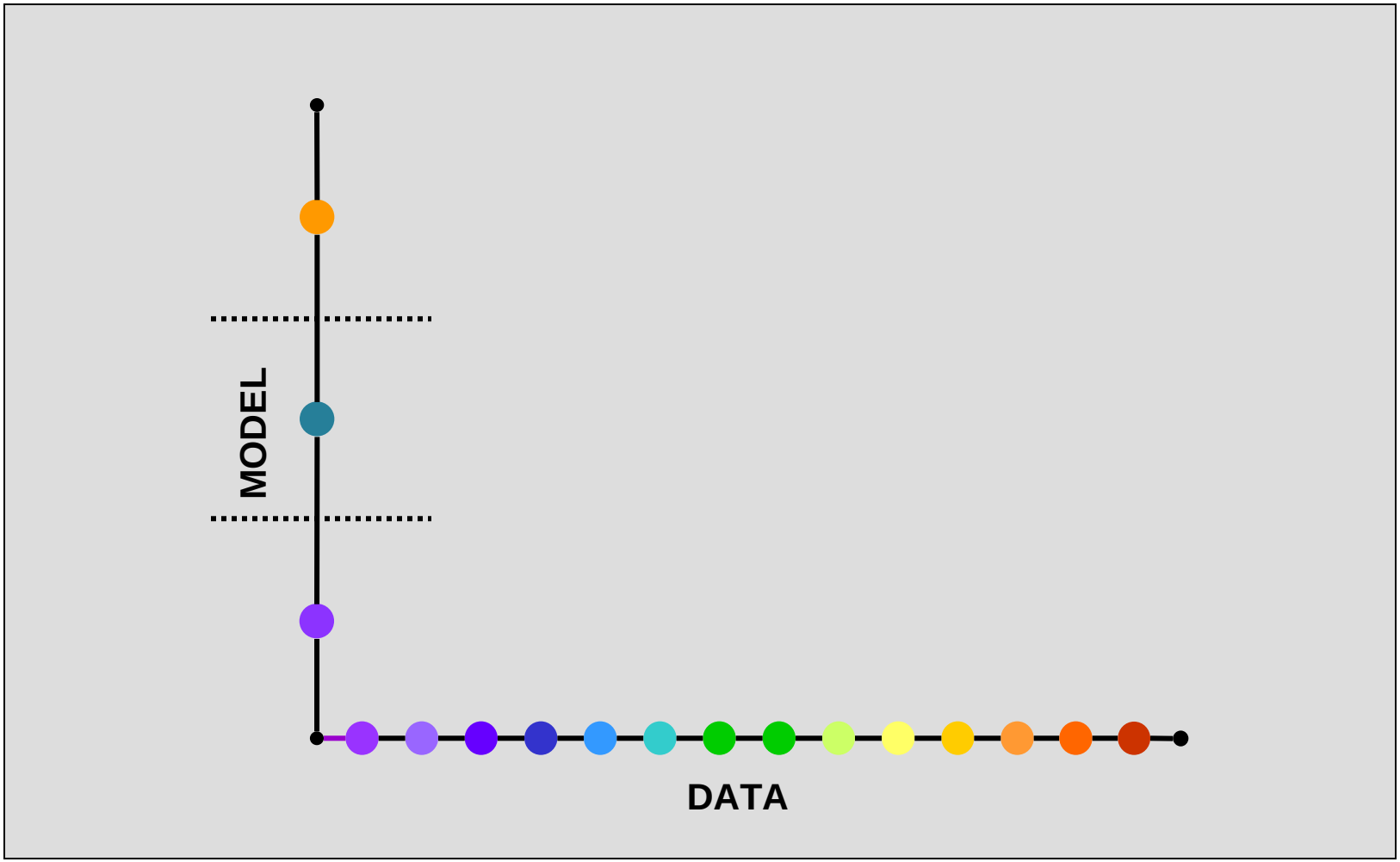
$$m_j = \frac{1}{N_j} \sum_{i \in segment(j)} v(i)$$

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

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

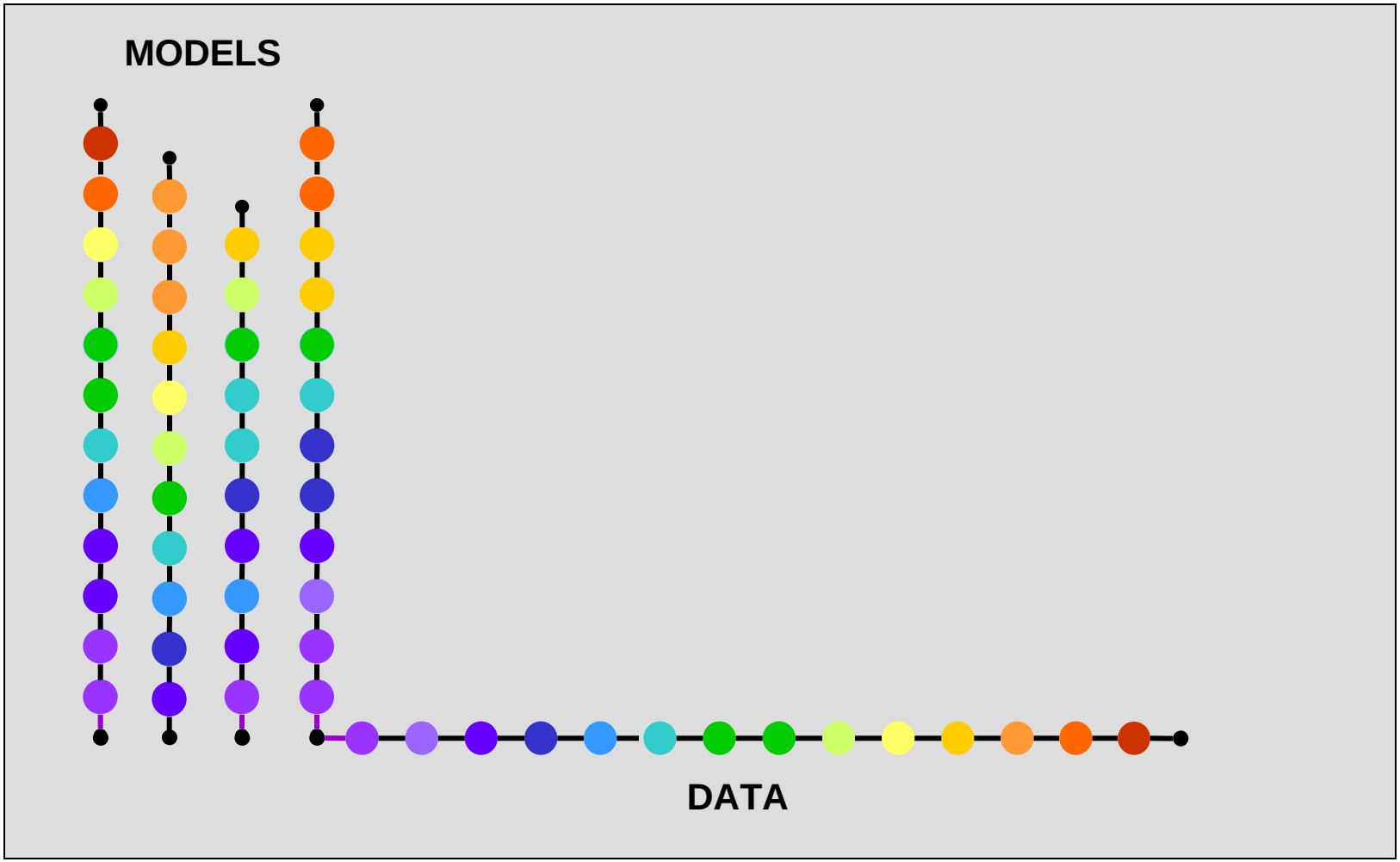
$v(i)$ is the i^{th} training vector

DTW with one model



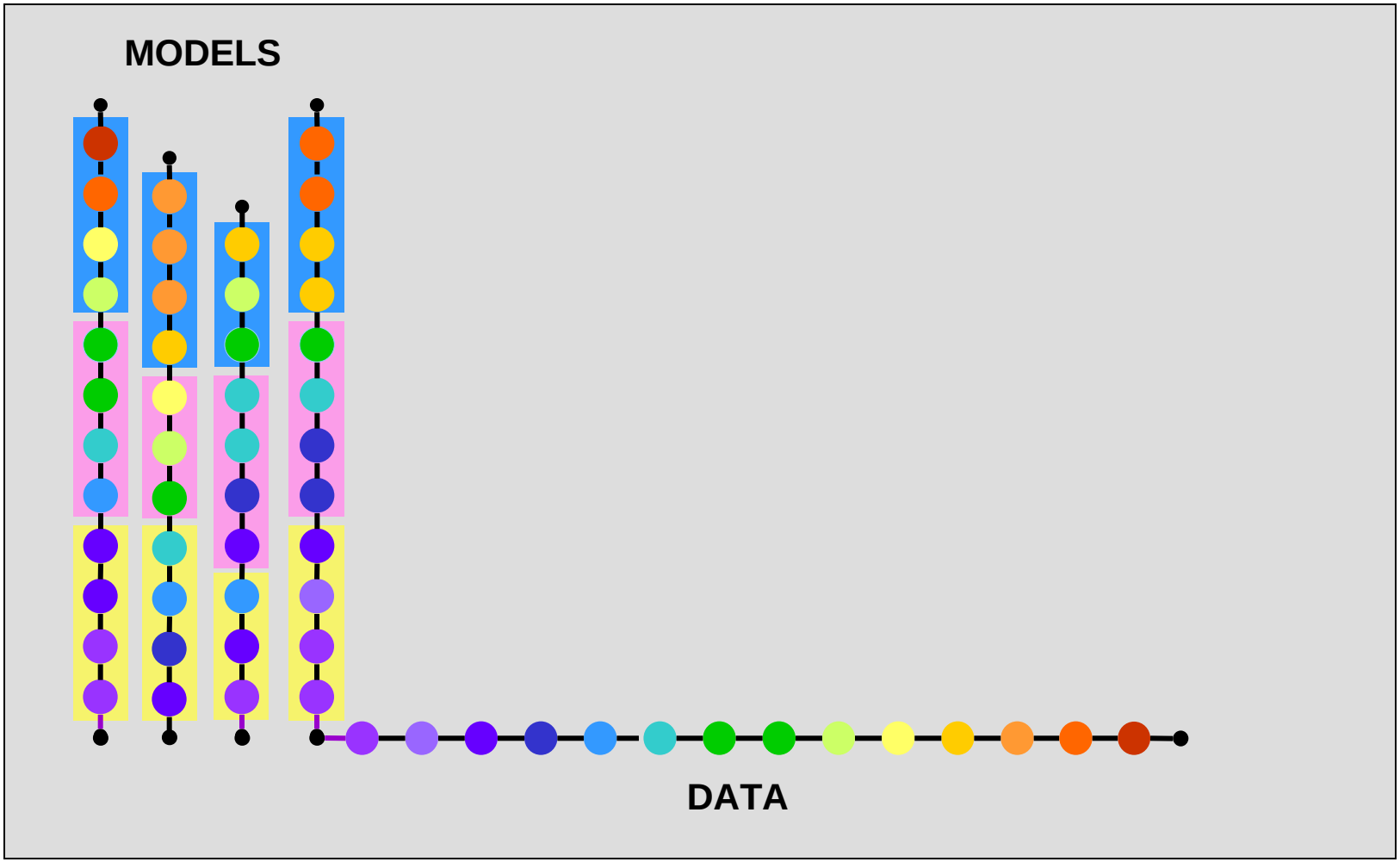
The averaged template is matched against the data string to be recognized
Select the word whose averaed template has the lowest cost of match

DTW with multiple models



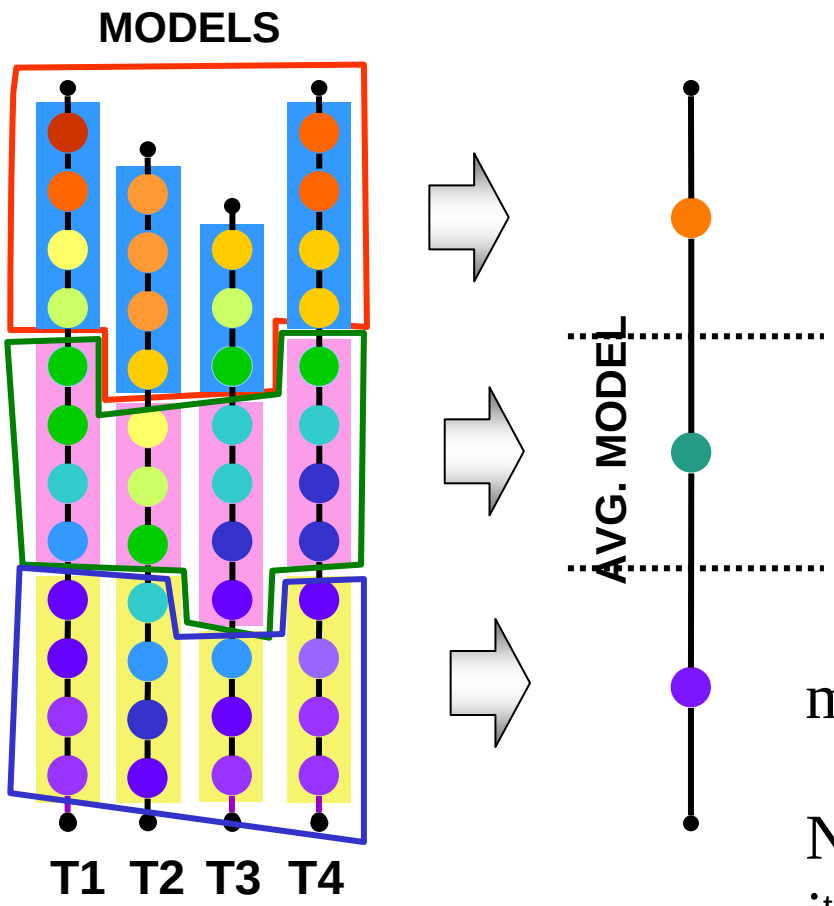
Segment all templates
Average each region into a single point

DTW with multiple models



Segment all templates
Average each region into a single point

DTW with multiple models



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

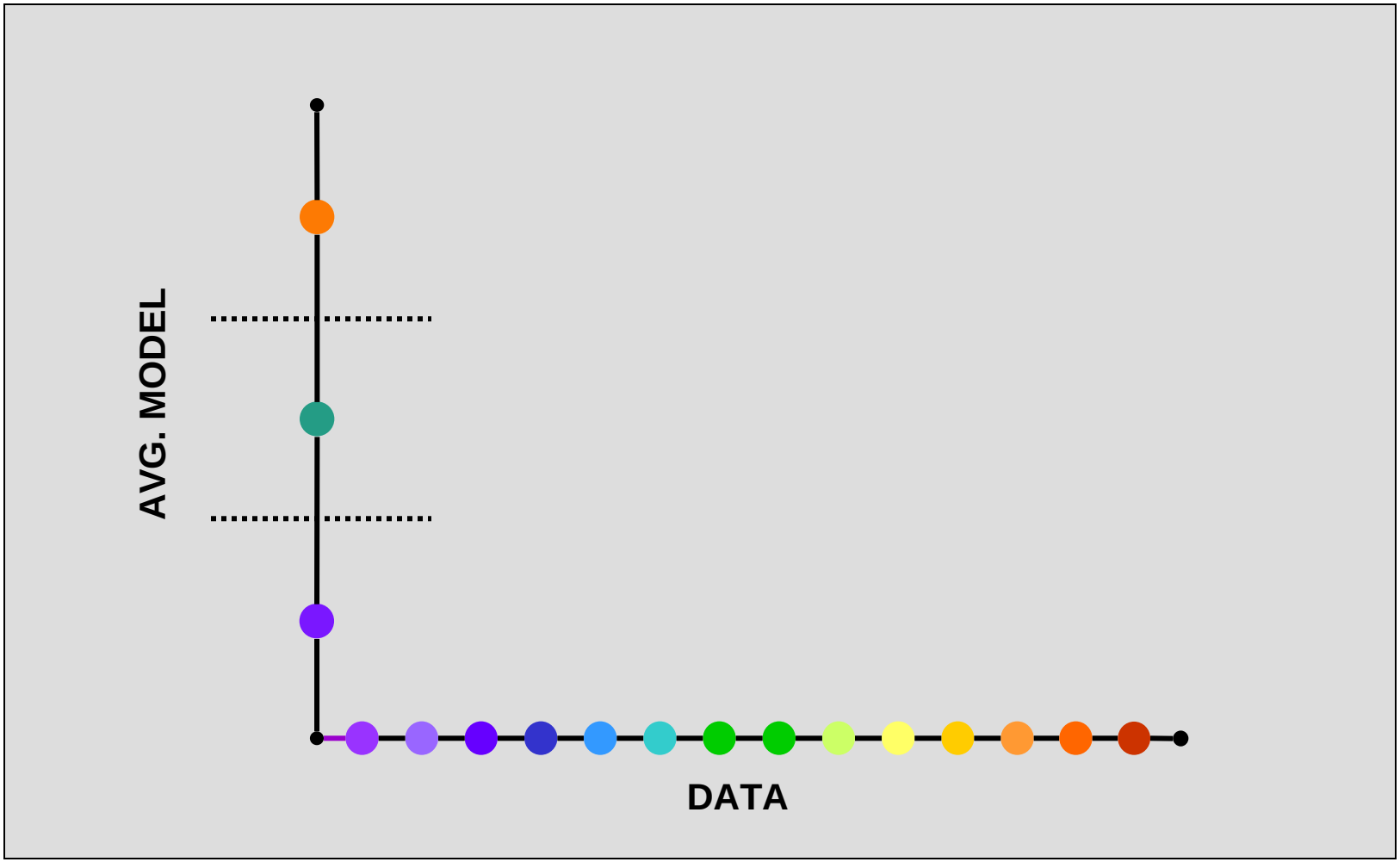
$segment_k(j)$ is the j^{th} segment of the k^{th} training sequence

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

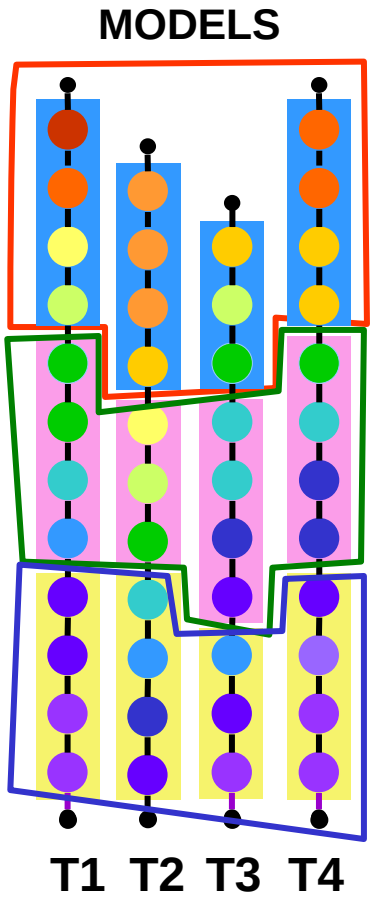
$v_k(i)$ is the i^{th} vector of the k^{th} training sequence

DTW with multiple models



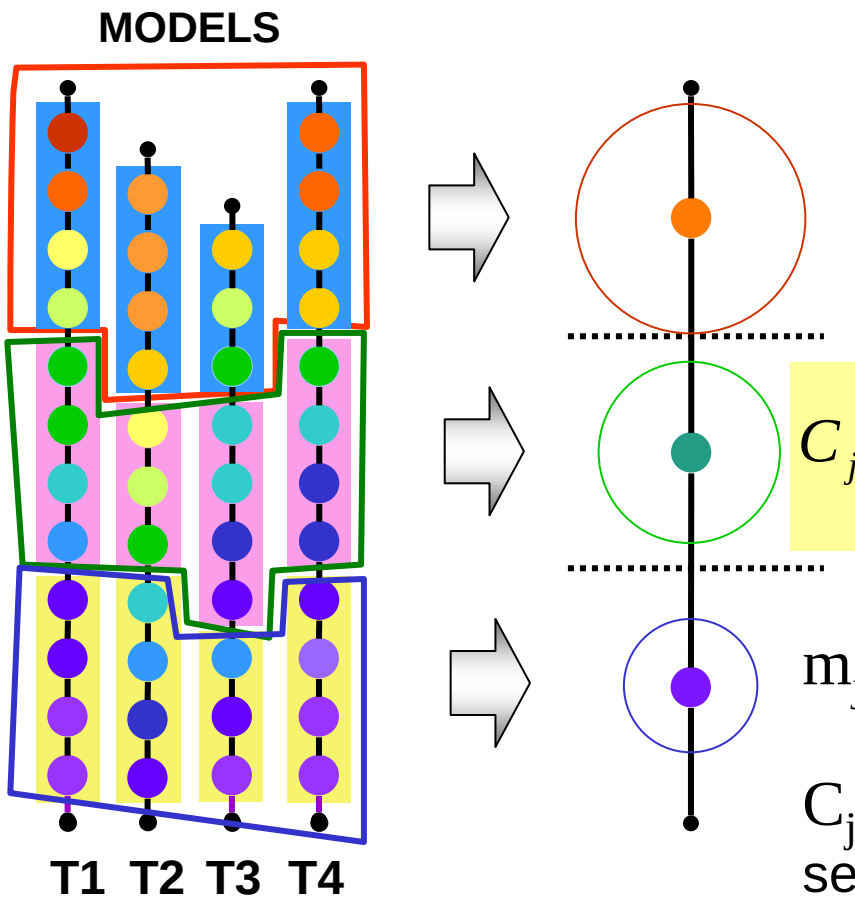
Segment all templates
Average each region into a single point
To get a simple average model, which is used for recognition

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

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

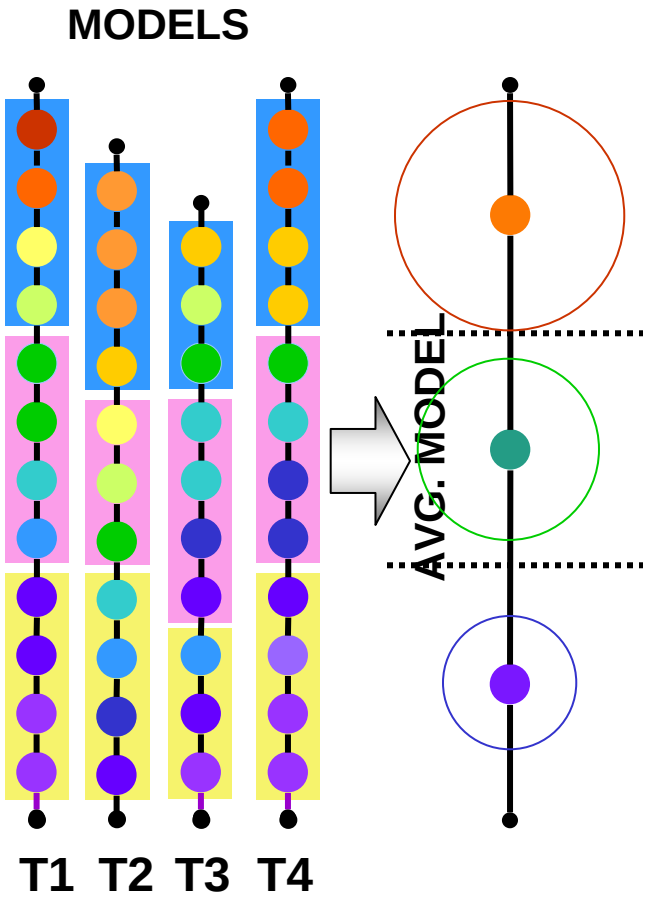
$$d(v,m_j)=-\log\left(Gaussian(v;m_j,C_j)\right)$$

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

Segmental K-means

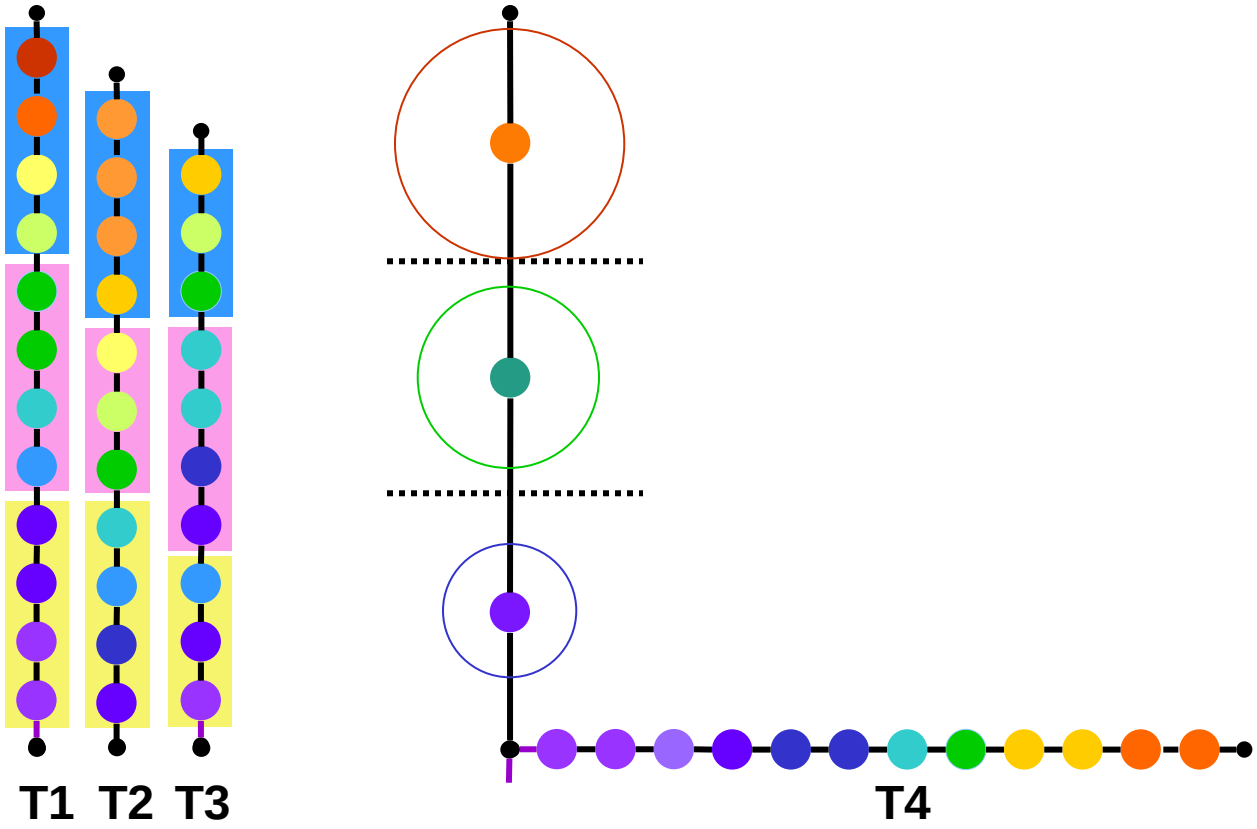
- Simple uniform segmentation of training instances is not the most effective method of grouping vectors in the training sequences
- A better segmentation strategy is to segment the training sequences such that the vectors within any segment are most alike
  - The total distance of vectors within each segment from the model vector for that segment is minimum
  - For a global optimum, the total distance of all vectors from the model for their respective segments must be 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



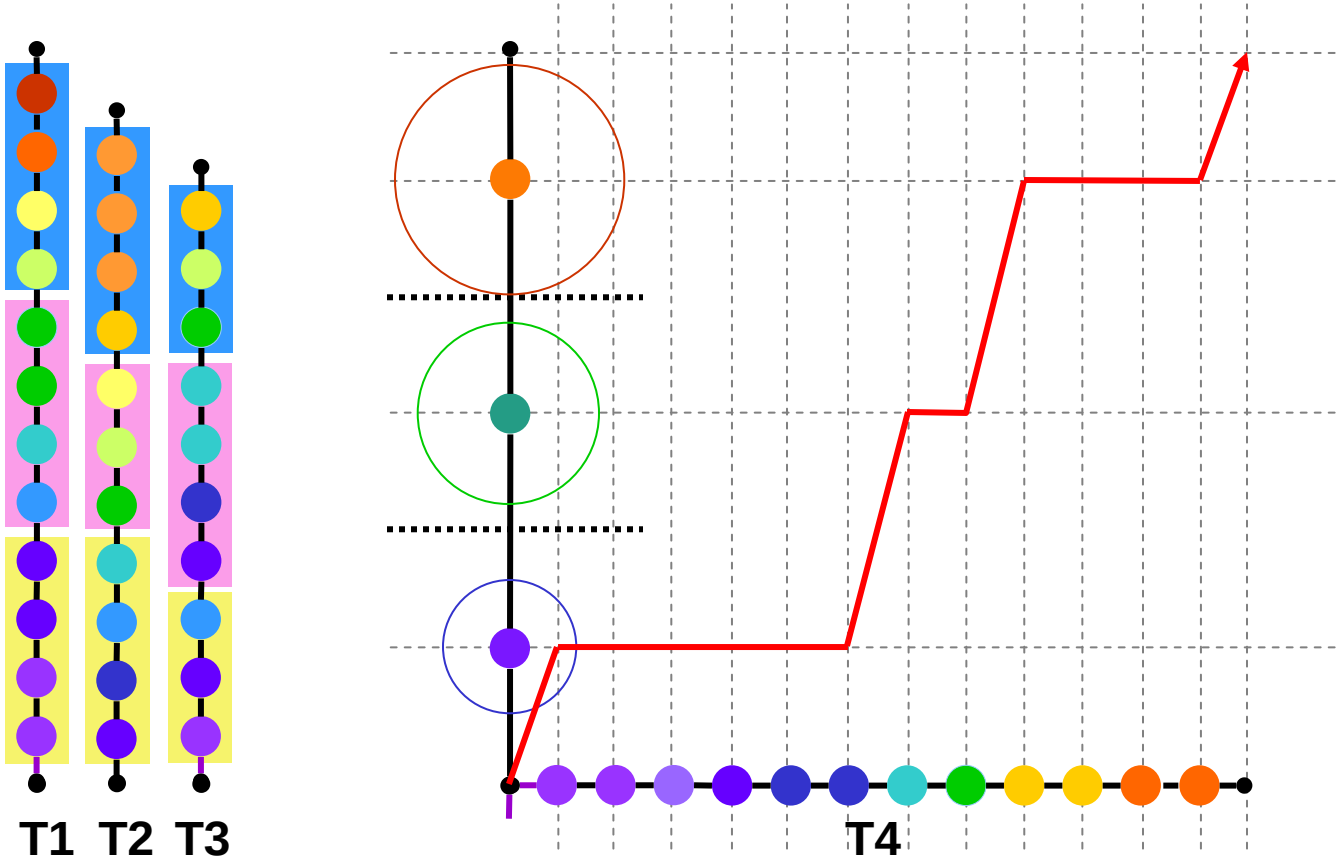
Initialize by uniform segmentation

Alignment for training a model from multiple vector sequences



Initialize by uniform segmentation

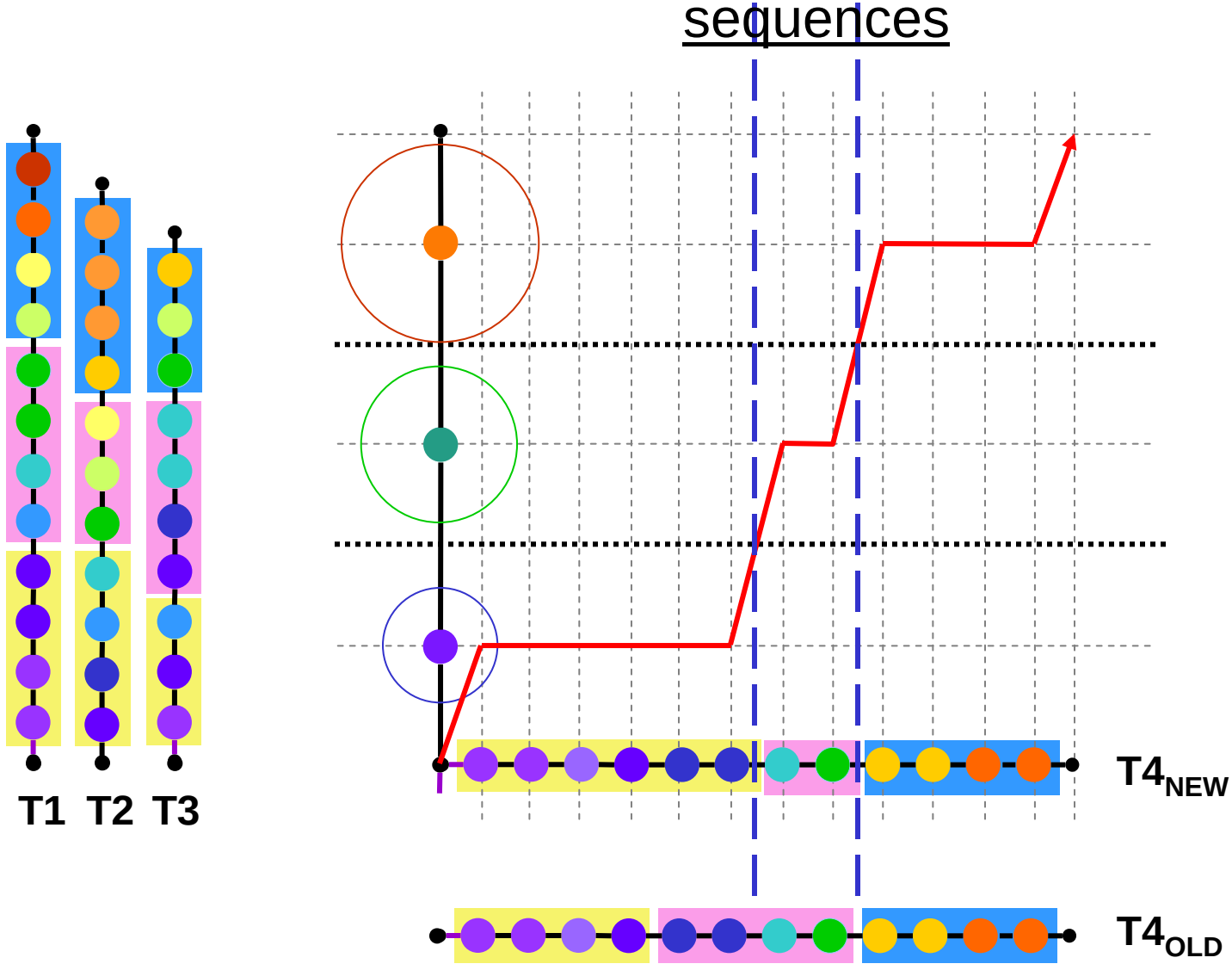
Alignment for training a model from multiple vector sequences



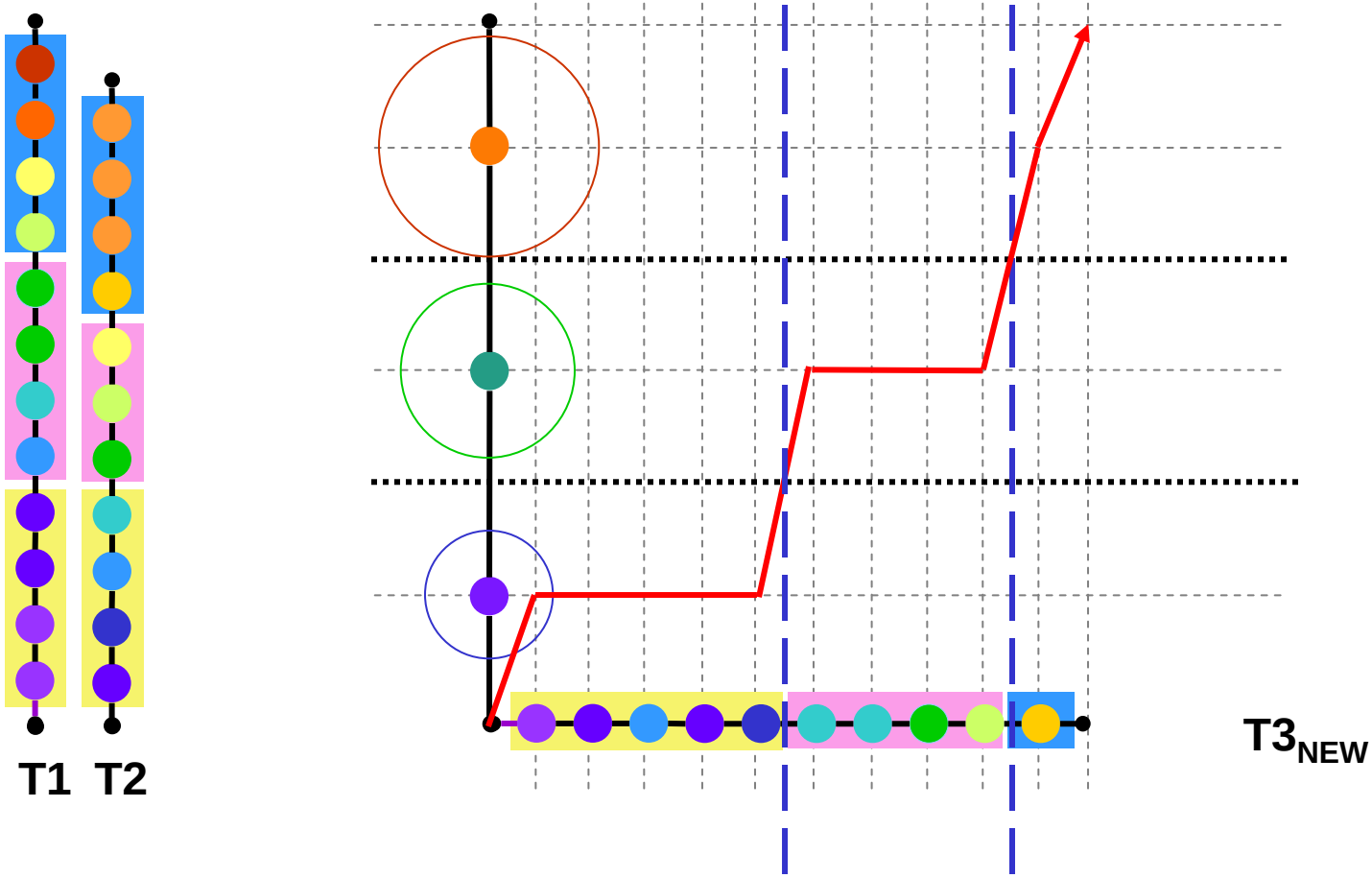
Initialize by uniform segmentation

Align each template to the averaged model to get new segmentations

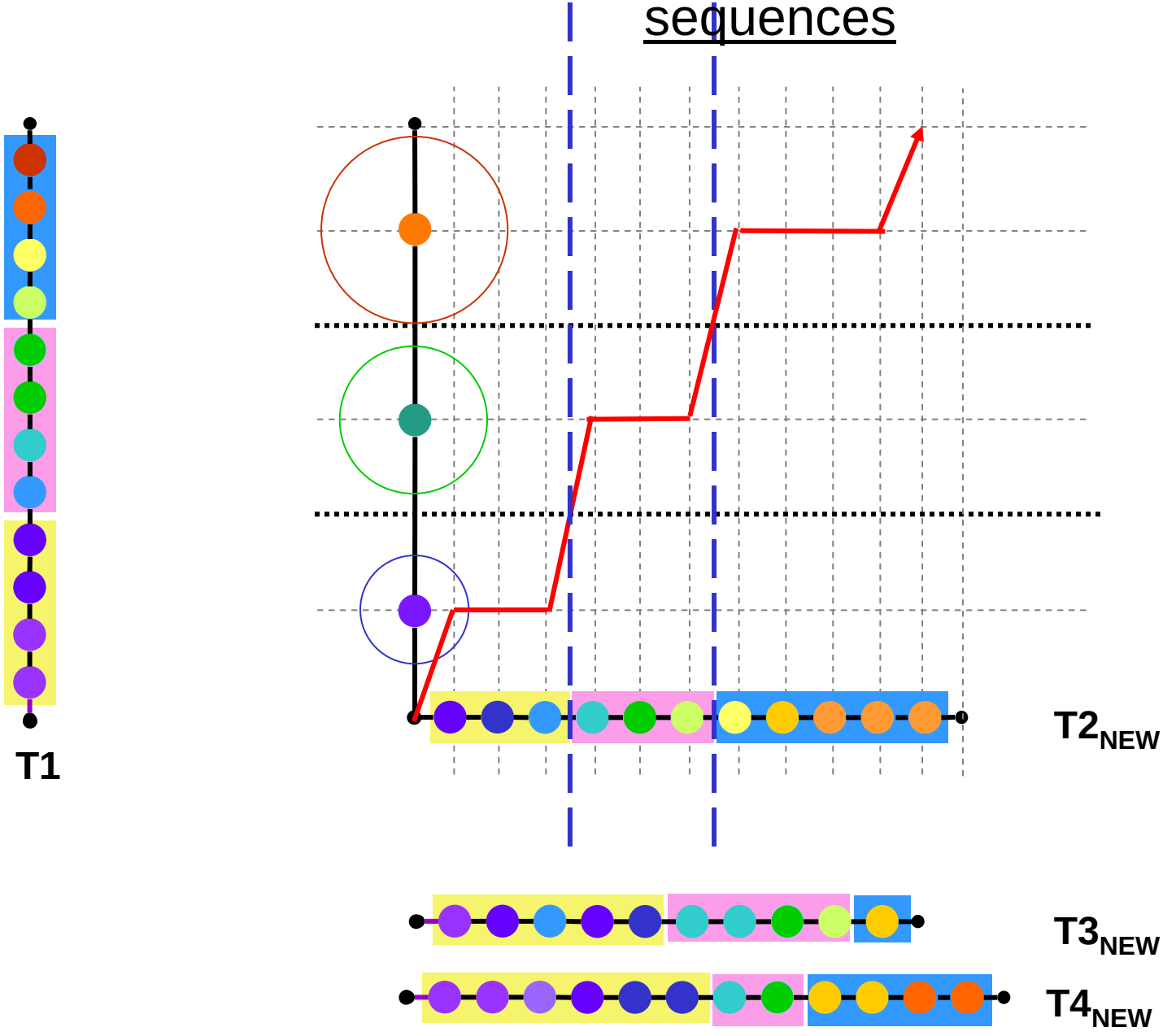
Alignment for training a model from multiple vector sequences



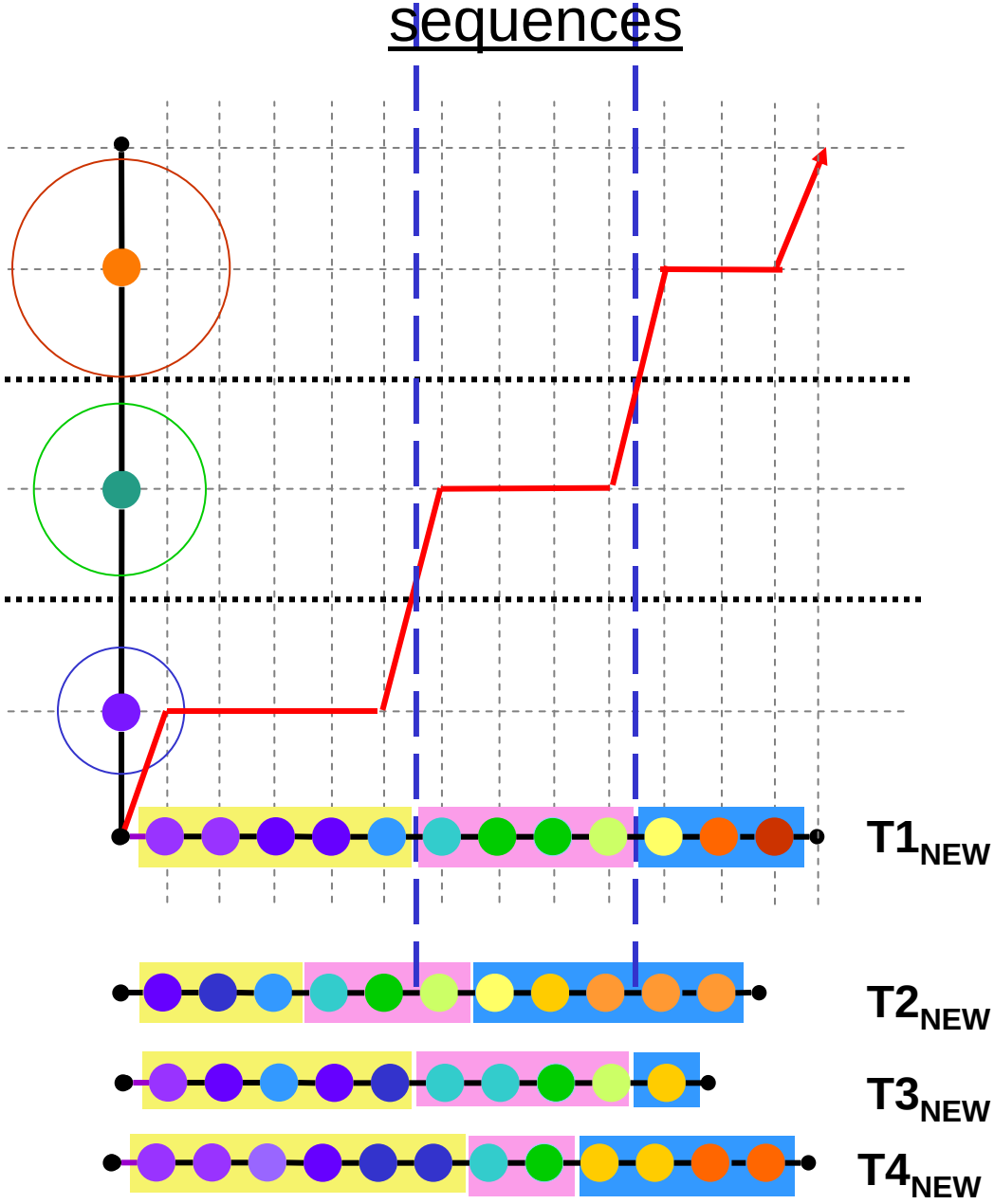
Alignment for training a model from multiple vector sequences



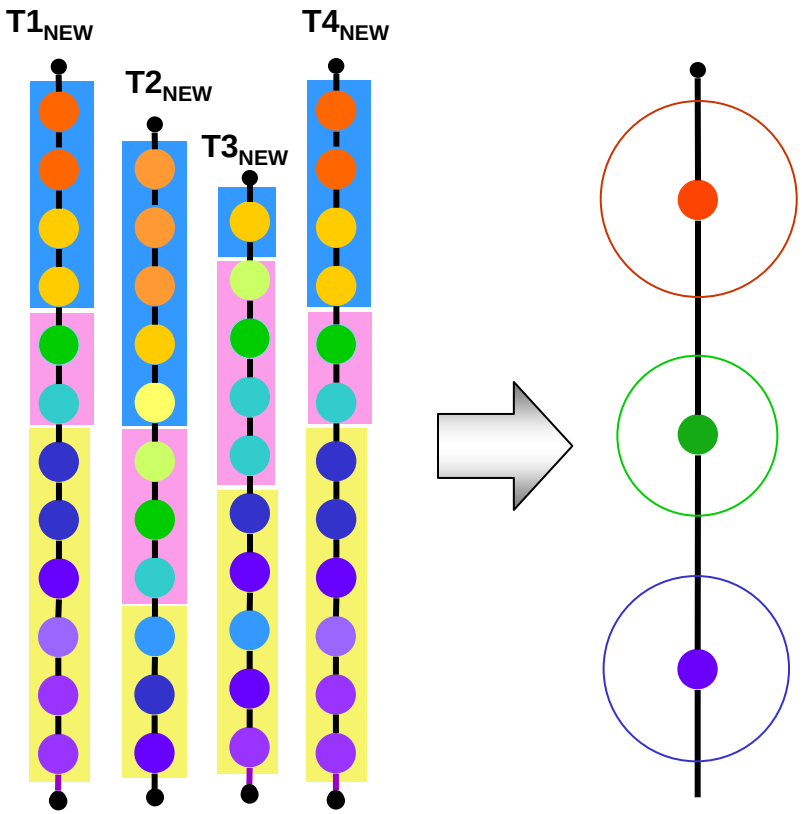
Alignment for training a model from multiple vector sequences



Alignment for training a model from multiple vector sequences

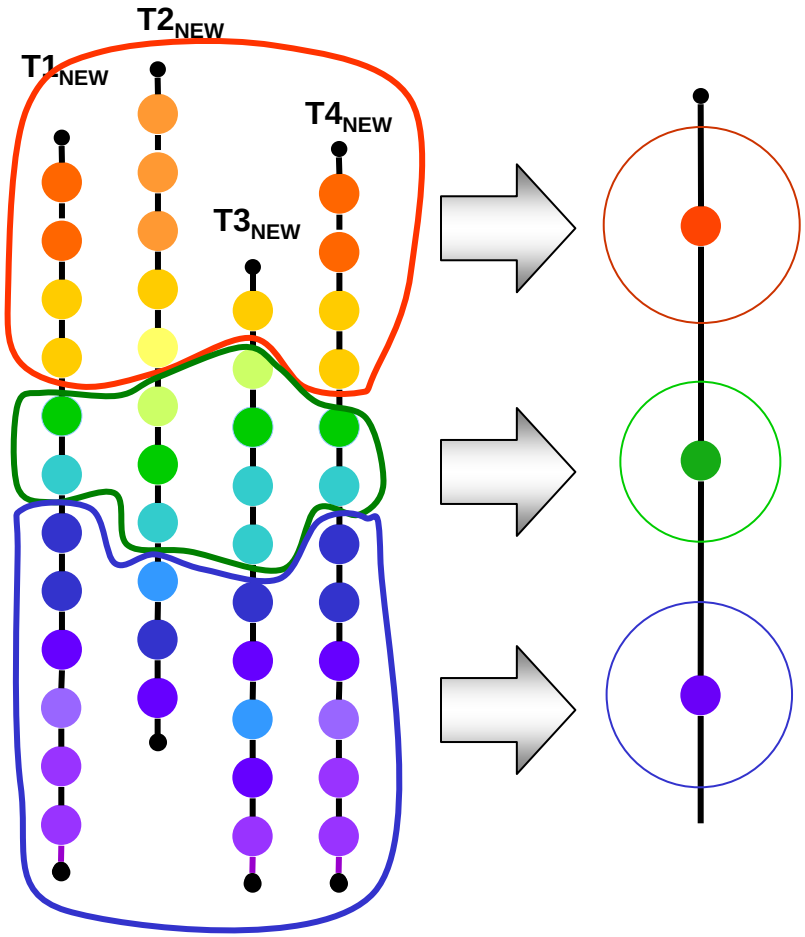


Alignment for training a model from multiple vector sequences

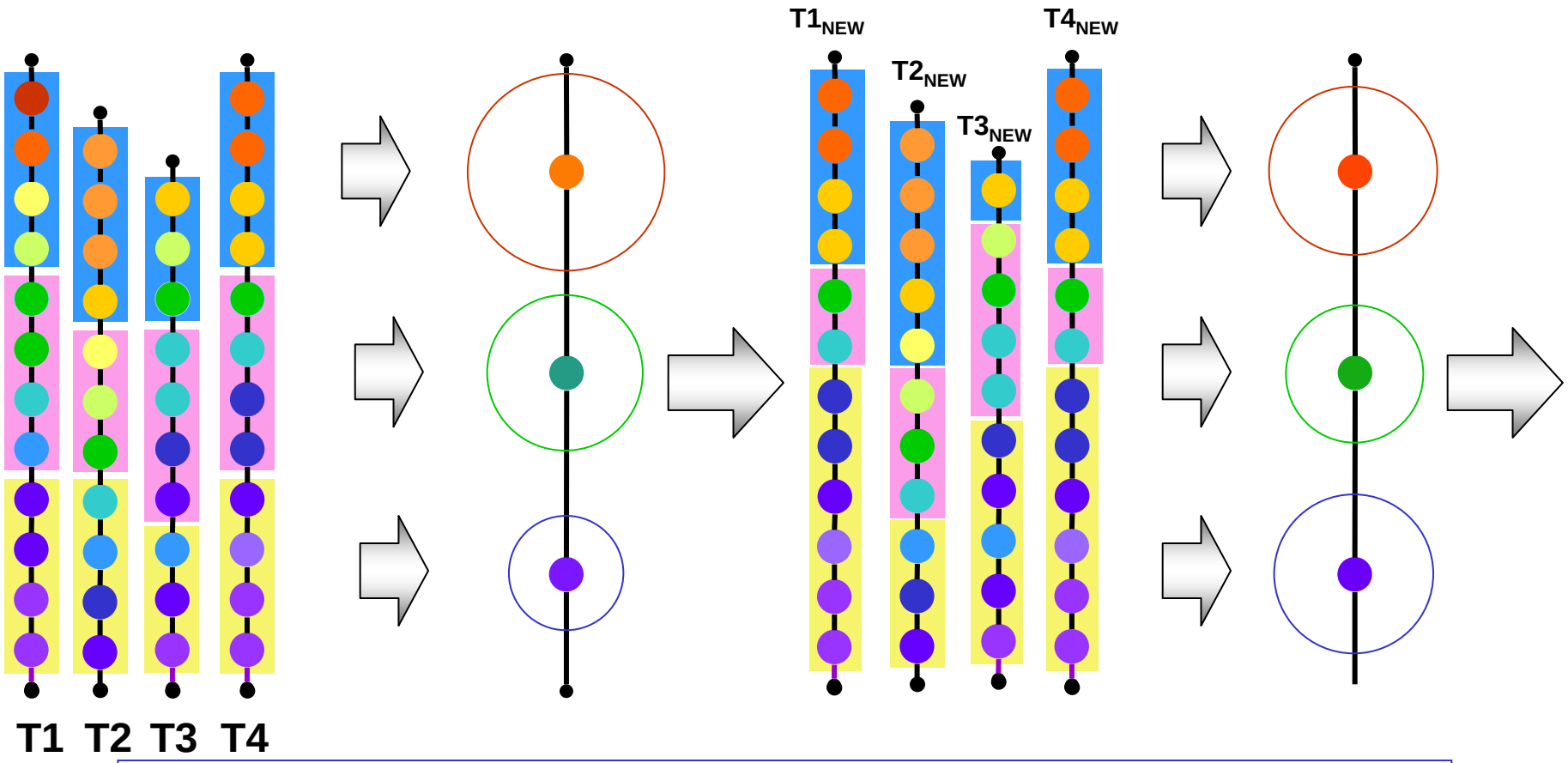


Initialize by uniform segmentation
Align each template to the averaged model to get new segmentations
Recompute the average model from new segmentations

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

DTW with multiple models

- The variance formula just shown will result in very poor estimates for the covariance if the number of training vectors in a segment is small
  - If a segment has only 1 vector, the variance is 0!
- In practice, when the number of training vectors is small, the variance estimate is improved by sharing or smoothing
  - Example of sharing: all segments have the same *grand* covariance:

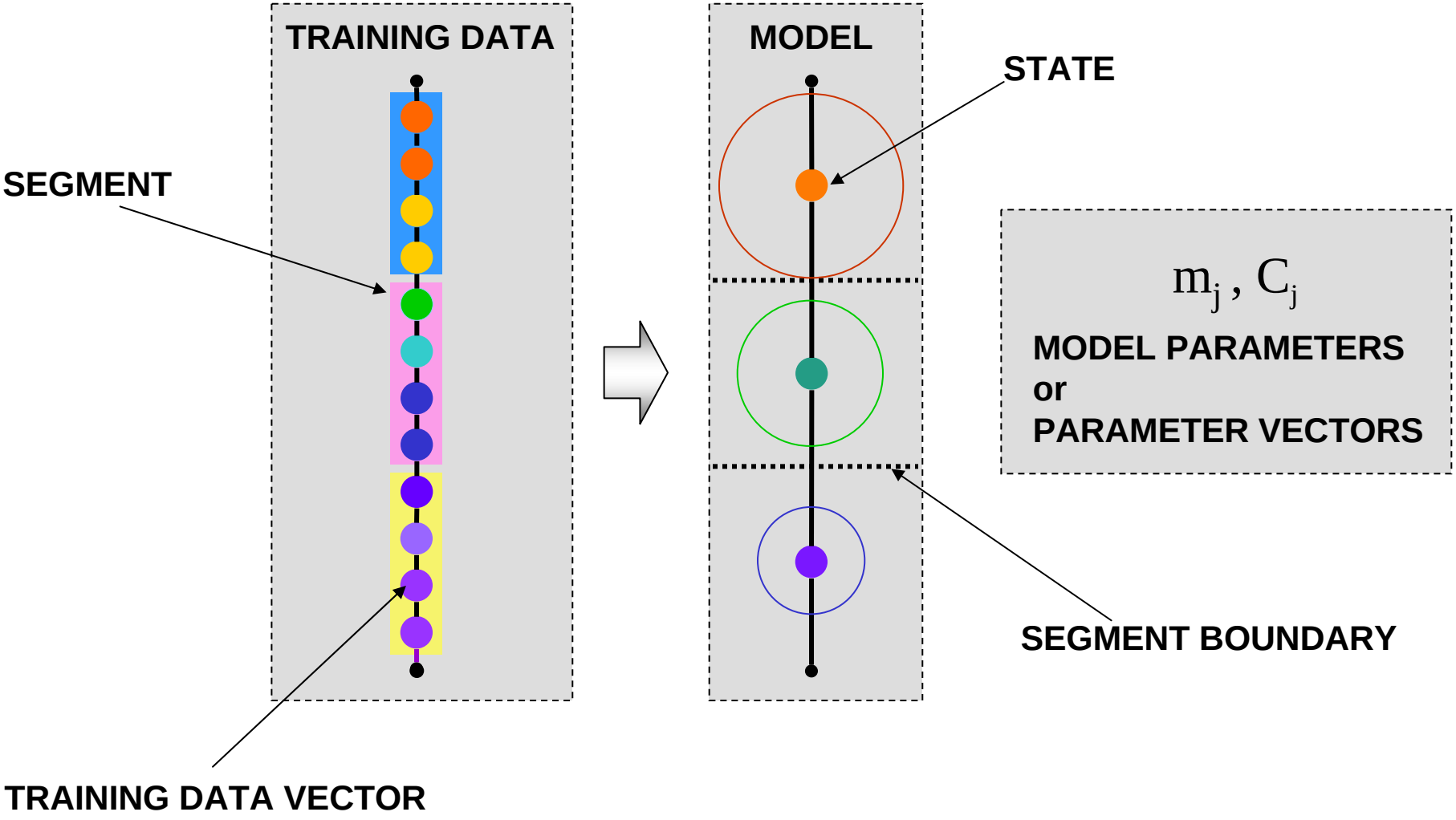
$$C_j = \frac{1}{\sum_k N_k} \sum_k \sum_{i \in segment_k(j)} \left(v_k(i) - m_j\right)\left(v_k(i) - m_j\right)^T$$

    - N_k is the number of vectors in the kth model sequence
  - Example of smoothing: interpolate between segment specific covariance and the grand covariance :

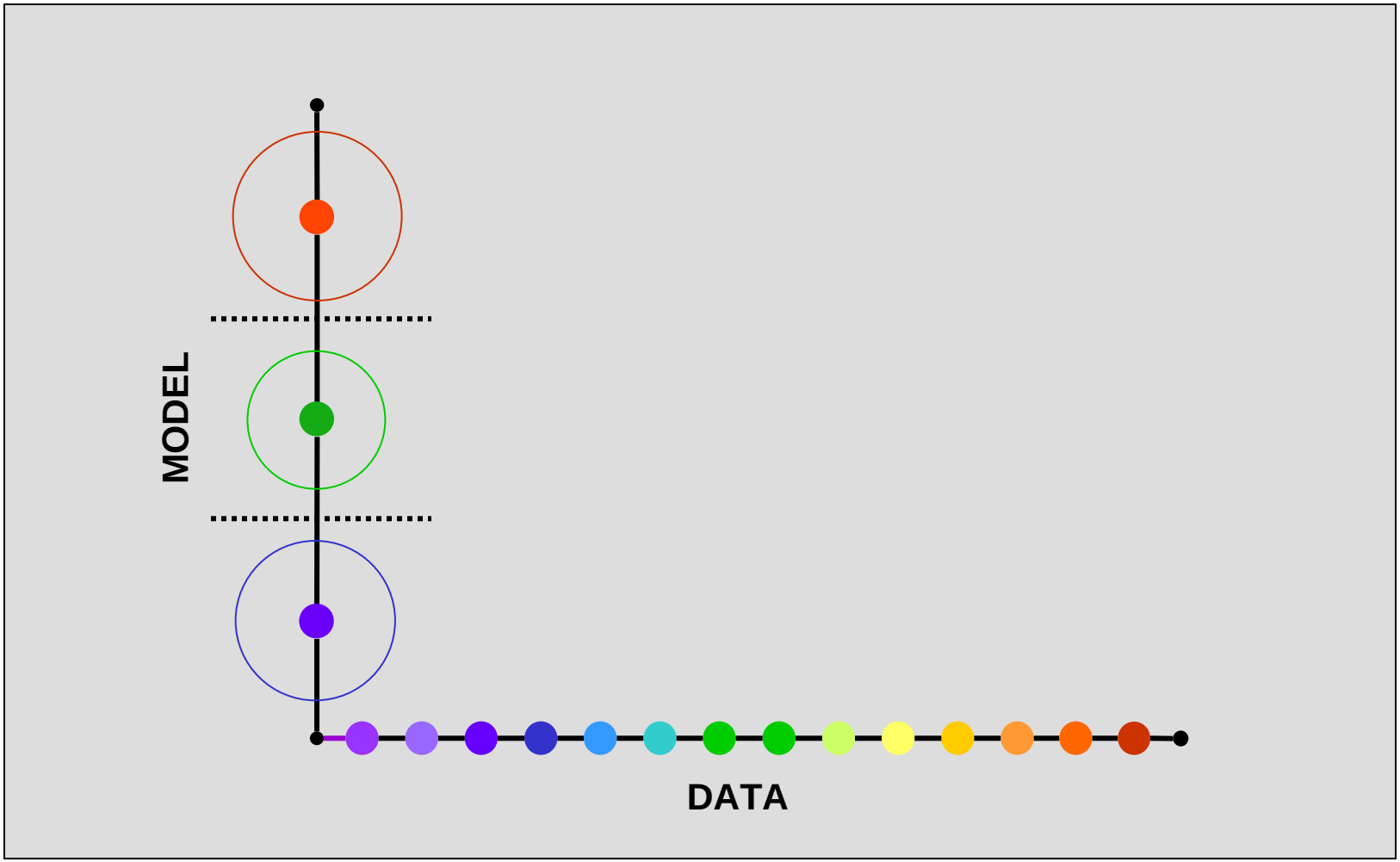
$$C_j = (1-\alpha)\frac{1}{\sum_k N_k} \sum_k \sum_{i \in segment_k(j)} \left(v_k(i) - m_j\right)\left(v_k(i) - m_j\right)^T + \alpha \frac{1}{\sum_k N_{k,j}} \sum_{i \in segment_k(j)} \left(v_k(i) - m_j\right)\left(v_k(i) - m_j\right)^T$$

    - A typical value of α is 0.5
- There are also other methods of estimating covariances more robustly

Shifted terminology



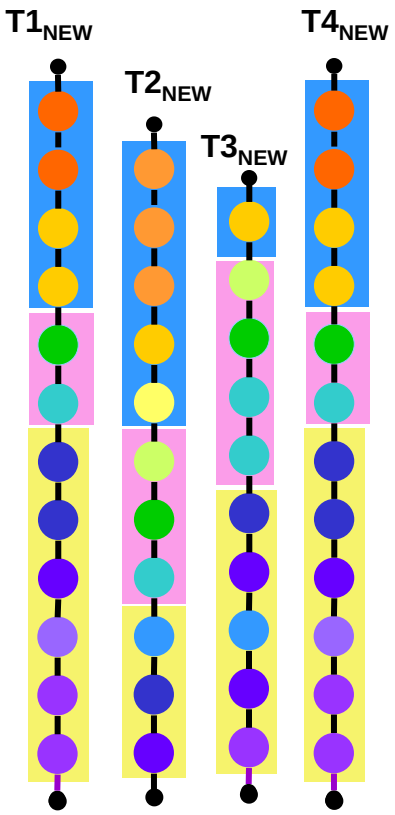
Transition structures in models



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

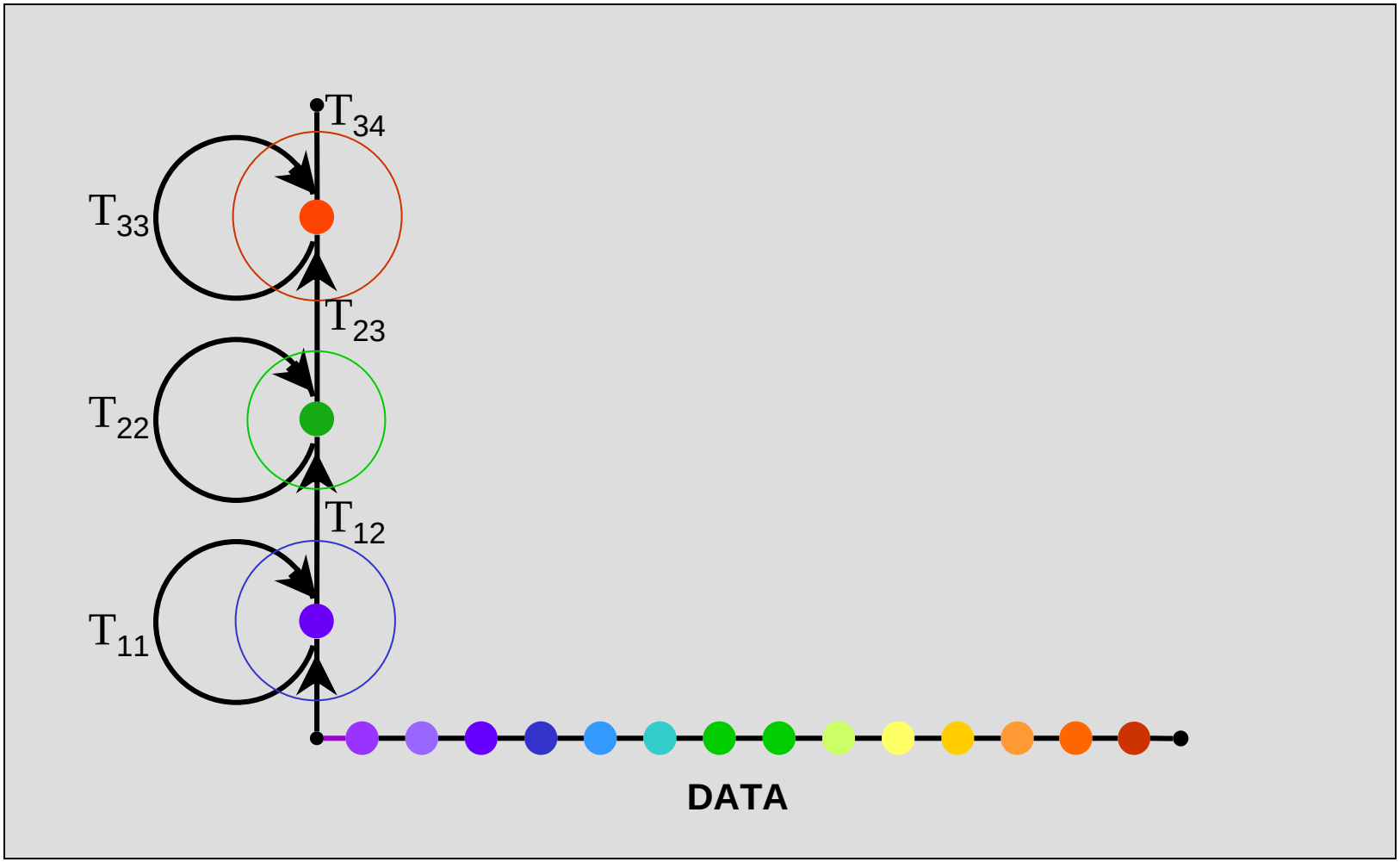
Model structure in incomplete.

DTW with multiple models



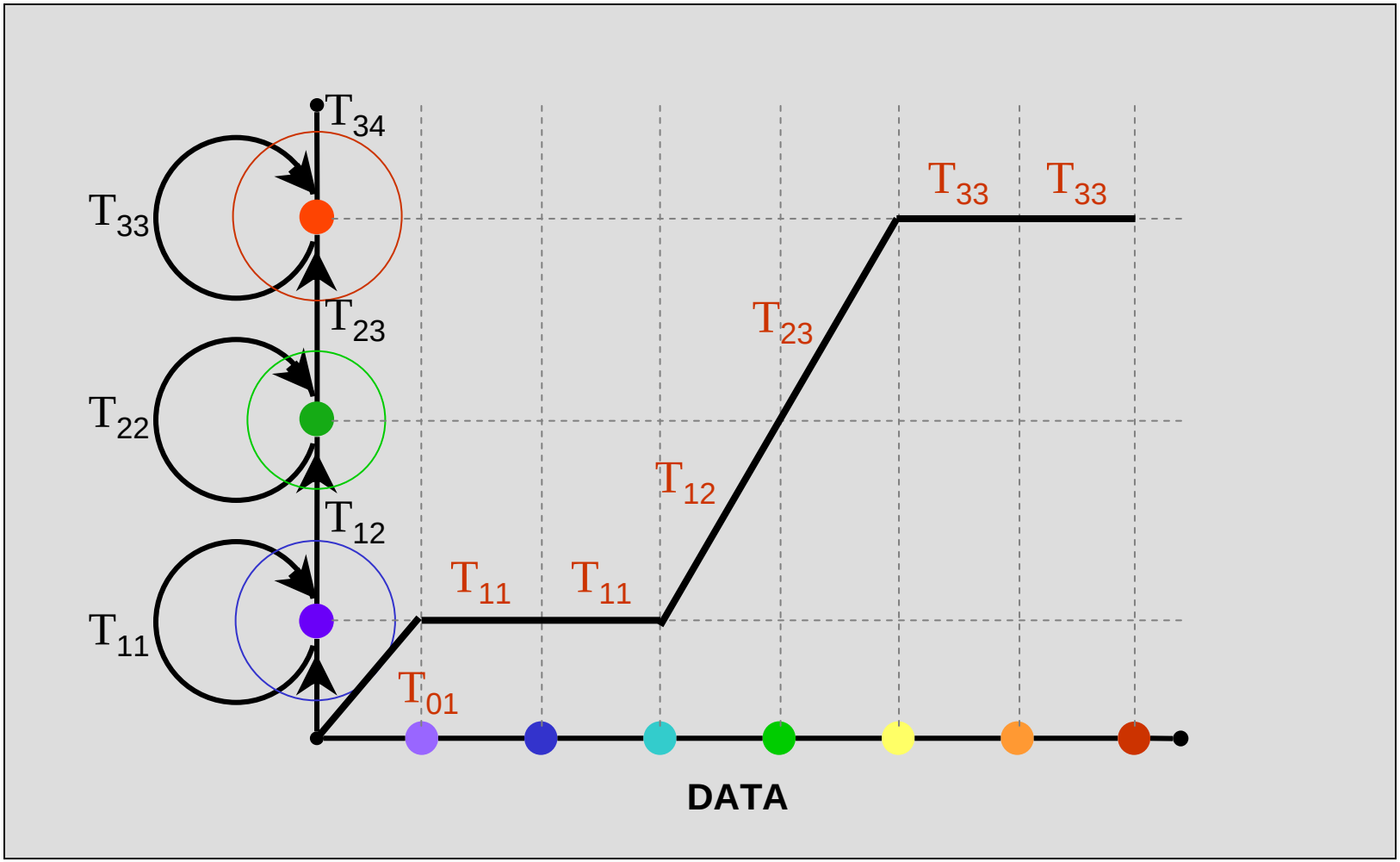
- Some segments are naturally longer than others
  - E.g., in the example the initial (yellow) segments are usually longer than the second (pink) segments
- This difference in segment lengths is different from the *variation* within a segment
  - Segments with small variance could still persist very 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



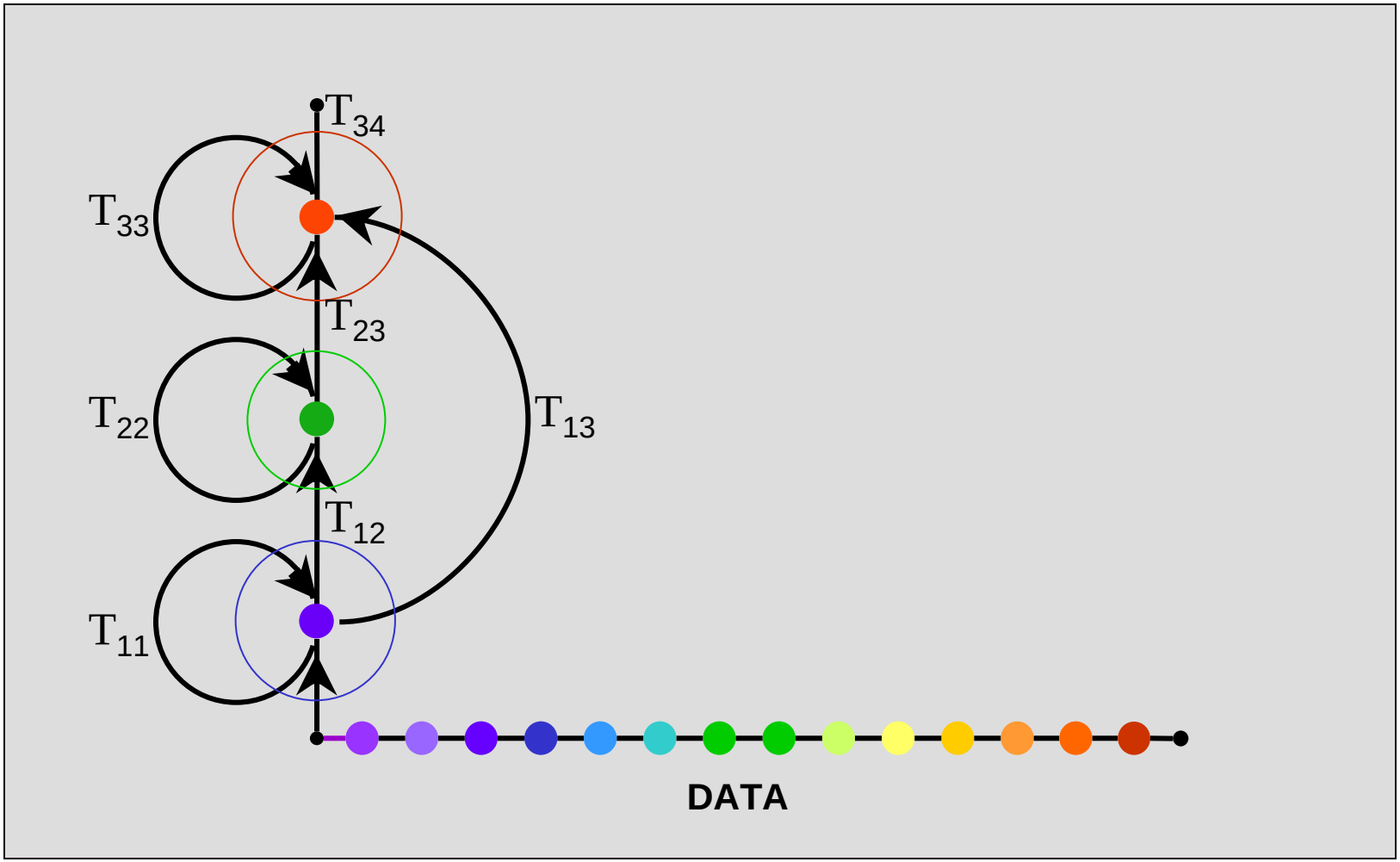
State specific insertion penalties are represented as self transition arcs for model vectors. Horizontal edges within the trellis will incur a penalty associated with the corresponding arc. Every transition within the model can have its own penalty.

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



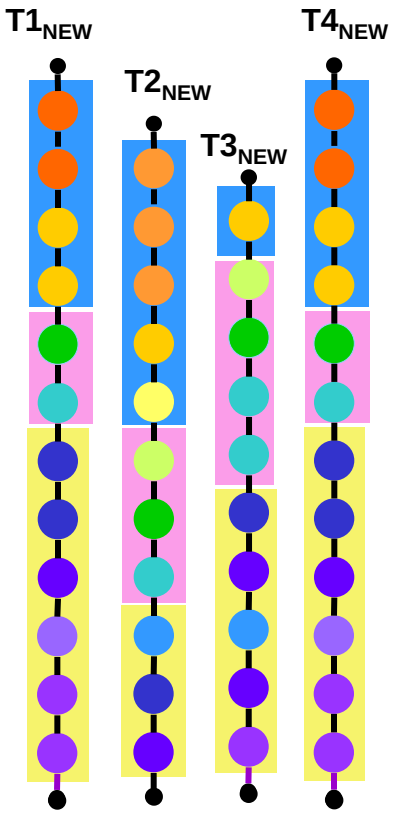
This structure also allows the inclusion of arcs that permit the central state to be skipped (deleted)

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
  - If the i^{th} segment is typically followed by the j^{th} segment, but also rarely by the k^{th} segment, then, if a data vector is within the i^{th} segment, the probability that the next data vector lies in the j^{th} segment is greater than the probability that it lies in the k^{th} segment
- A good choice for transition scores are the negative logarithm of the probabilities of the appropriate transitions
  - T_{ii} is the negative of the log of the probability that if the current data vector belongs to the i^{th} state, the next data vector will also belong to the i^{th} state
  - T_{ij} is the negative of the log of the 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

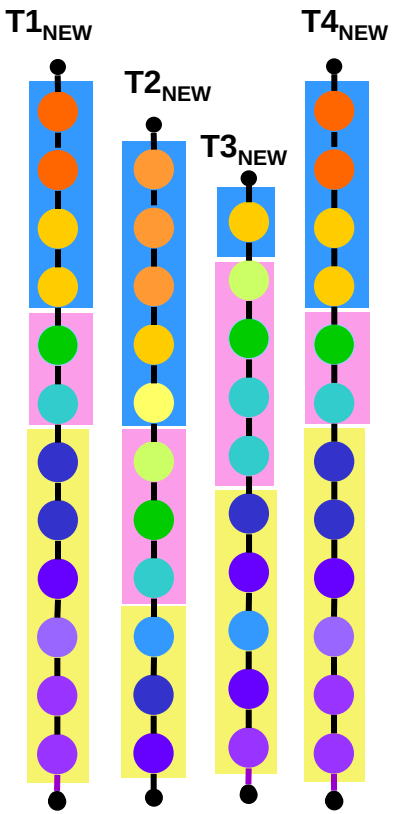


- Transition scores can be easily computed by a simple extension of the segmental K-means algorithm
- Probabilities can be counted by simple counting

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

- $N_{k,i}$ 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

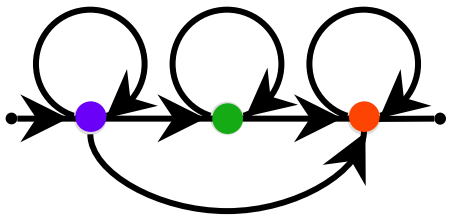
- A special score is the penalty associated with *starting* at a particular state
- In our examples we always begin at the first state
- Enforcing this is equivalent to setting $T_{01} = 0$, T_{0j} = infinity for j != 1
- It is sometimes useful to permit entry directly into later states
  - i.e. permit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

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

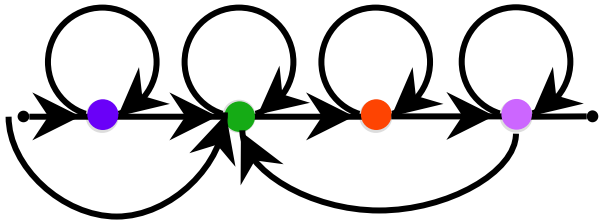
- N is the total number of training sequences
- N_{0j} is the number of training sequences for which the first data vector was in the j^{th} state

Modified segmental K-means AKA Viterbi training

- Some structural information must be prespecified
- The number of states must be prespecified
  - Various heuristics exist to determine this number automatically
  - Otherwise, the number must be manually specified
- 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

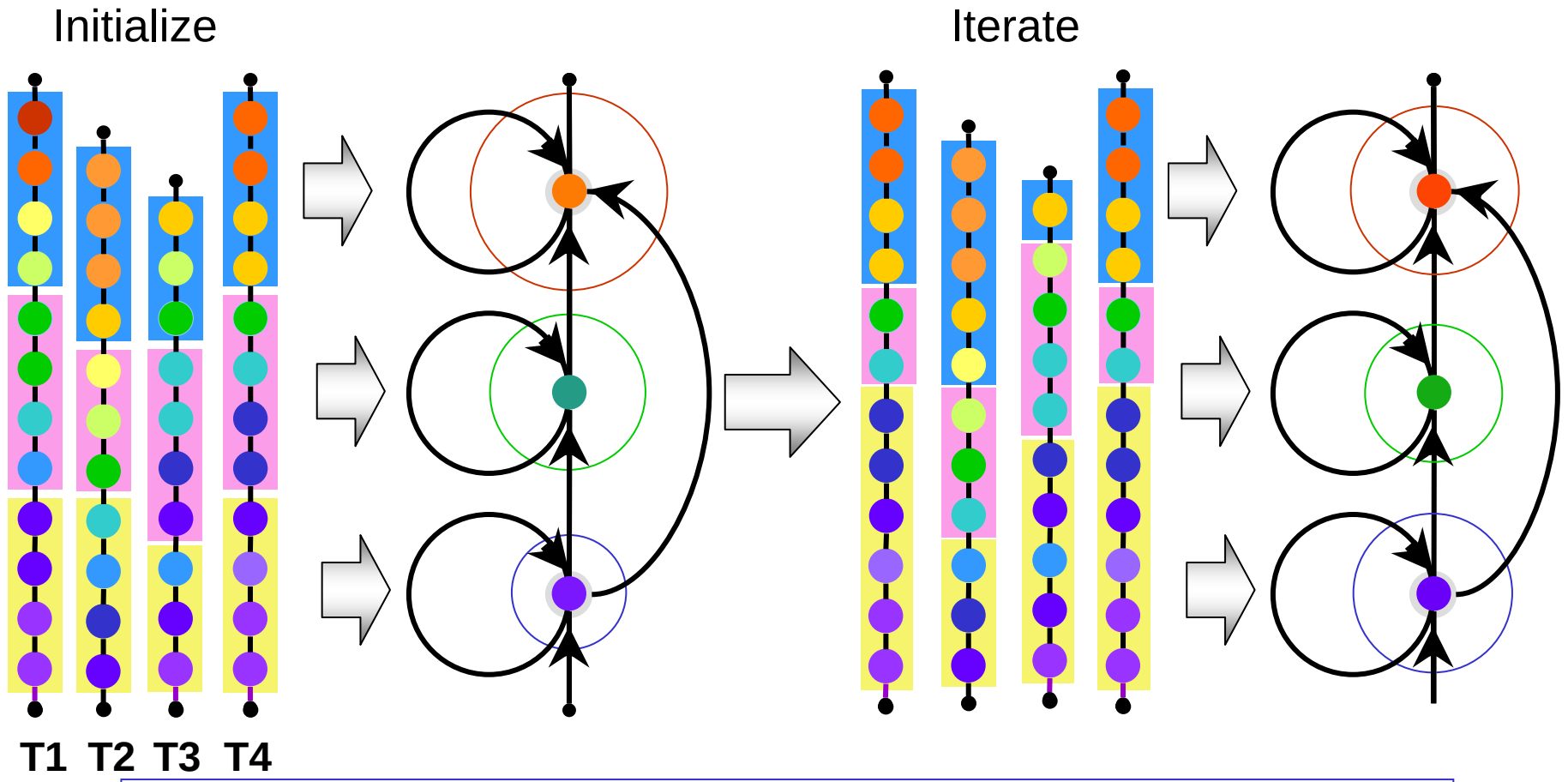
Modified segmental K-means AKA Viterbi training

- 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
  - You may choose to initialize parameters differently, e.g. by random values

Modified segmental K-means AKA Viterbi training

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

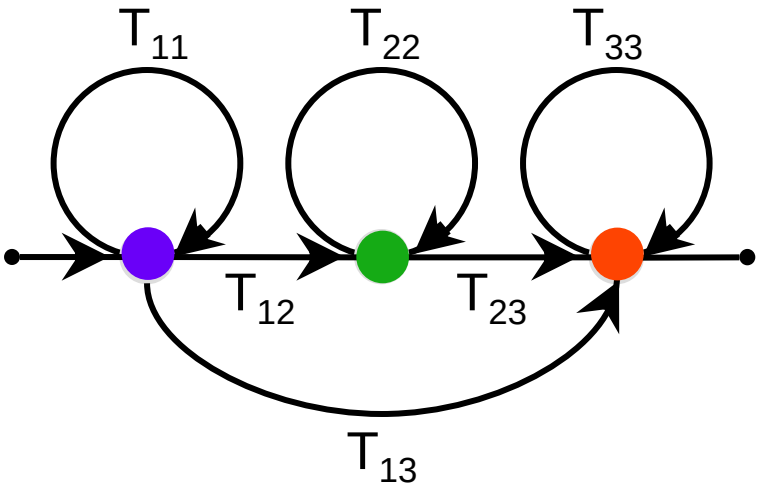
Alignment for training a model from multiple vector sequences



The procedure can be continued until convergence

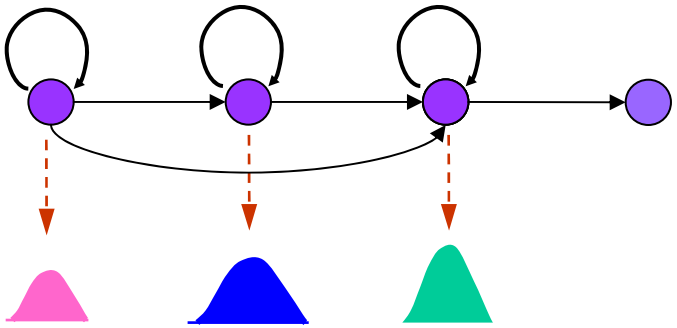
Convergence is achieved when the total best-alignment error for all training sequences does not change significantly with further refinement of the model

DTW and Hidden Markov Models (HMMs)

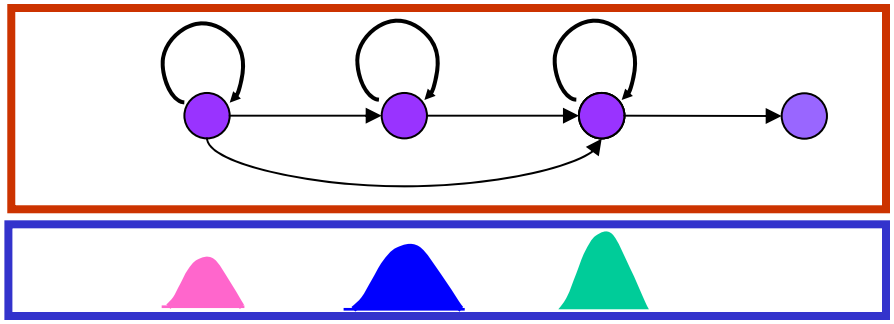


- This structure is a generic representation of a statistical model for processes that generate time series
- The "segments" in the time series are referred to as states
  - The process passes through these states to generate time series
- The entire structure may be viewed as *one* generalization of 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

Analogy between DTW vs. HMM

- DTW: Transition penalty HMM: Transition probability
  - The transition penalty of the DTW template is analogous to the negative *log* of the transition probability for the HMM
- DTW: Symbol matching cost HMM: State probability
  - The matching cost of DTW is analogous to the negative *log* of the probability of the observation computed from the probability distribution associated with the state
- DTW: minimizing cost HMM: Maximizing probability
- The string matching algorithm for DTW actually finds the sequence of states in the HMM that matches the observation

A change of notation

- Thus far we have been talking about *Costs*, that are in fact *Negative Log Probabilities*
- Henceforth we will talk in terms of Probabilities and not Log probabilities
  - A matter of convenience
  - This does not change the basic procedures – what used to be summation will now become multiplication
    - Ie. We multiply the probabilities along the best path, rather than to add them

QUESTIONS?

- ??