Joint Institute of Engineering

SUN YAT-SEN UNIVERSITY

Carnegie Mellon University

Design and Implementation of Speech Recognition Systems

Fall 2014

Ming Li

Class 6: Templates to HMMs

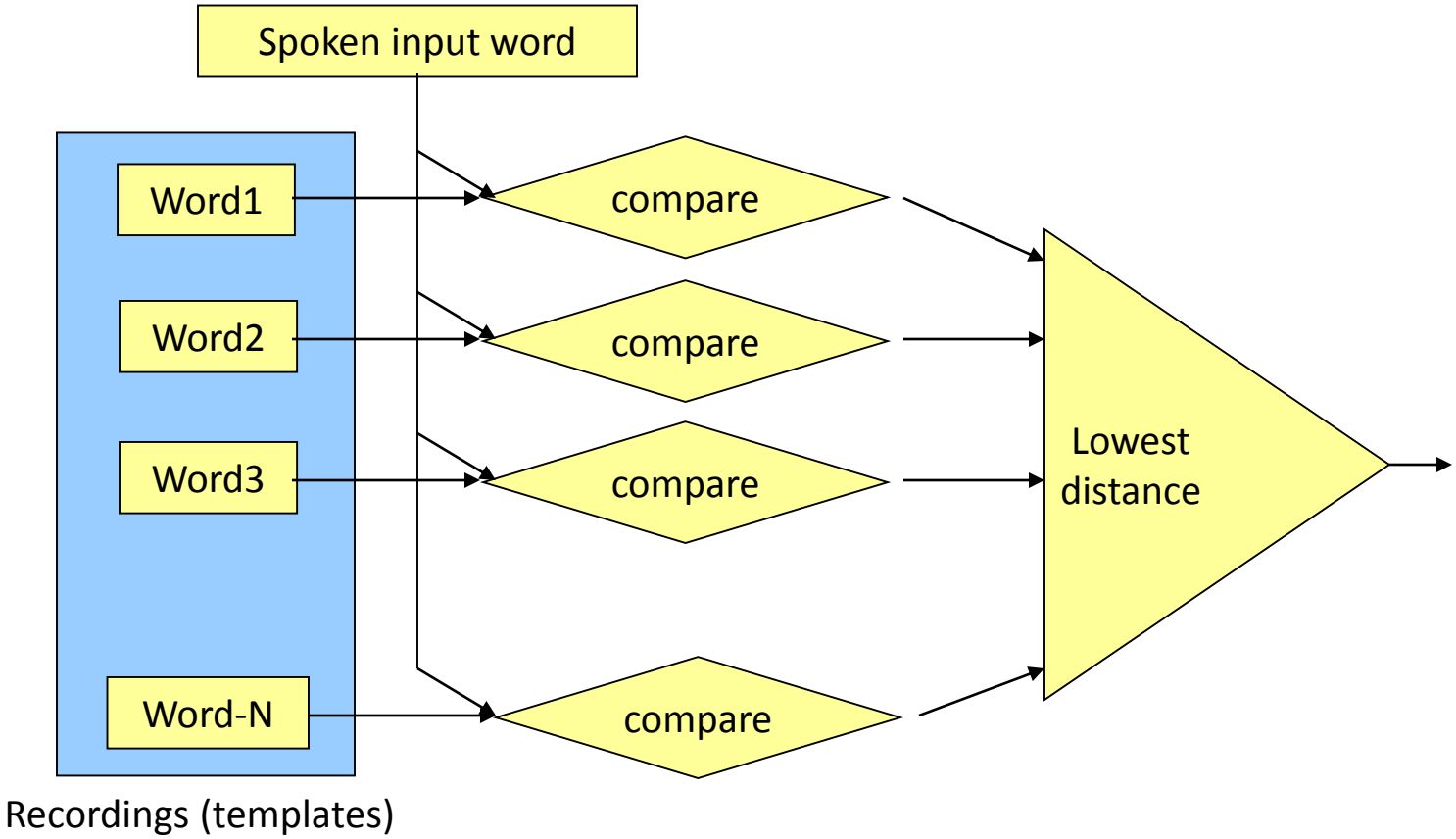
Oct 7th

Thanks to Professor Bhiksha Raj for the contribution of the slides

Recap

- Thus far, we have looked at dynamic programming for string matching,
- And derived DTW from DP for isolated word recognition
- We identified the search trellis, time-synchronous search as efficient mechanisms for decoding
- We looked at ways to improve search efficiency using pruning
  - In particular, we identified *beam pruning* as a nearly universal pruning mechanism in speech recognition
- We looked at the limitations of DTW and template matching:
  - Ok for limited, small vocabulary applications
  - Brittle; breaks down if speakers change

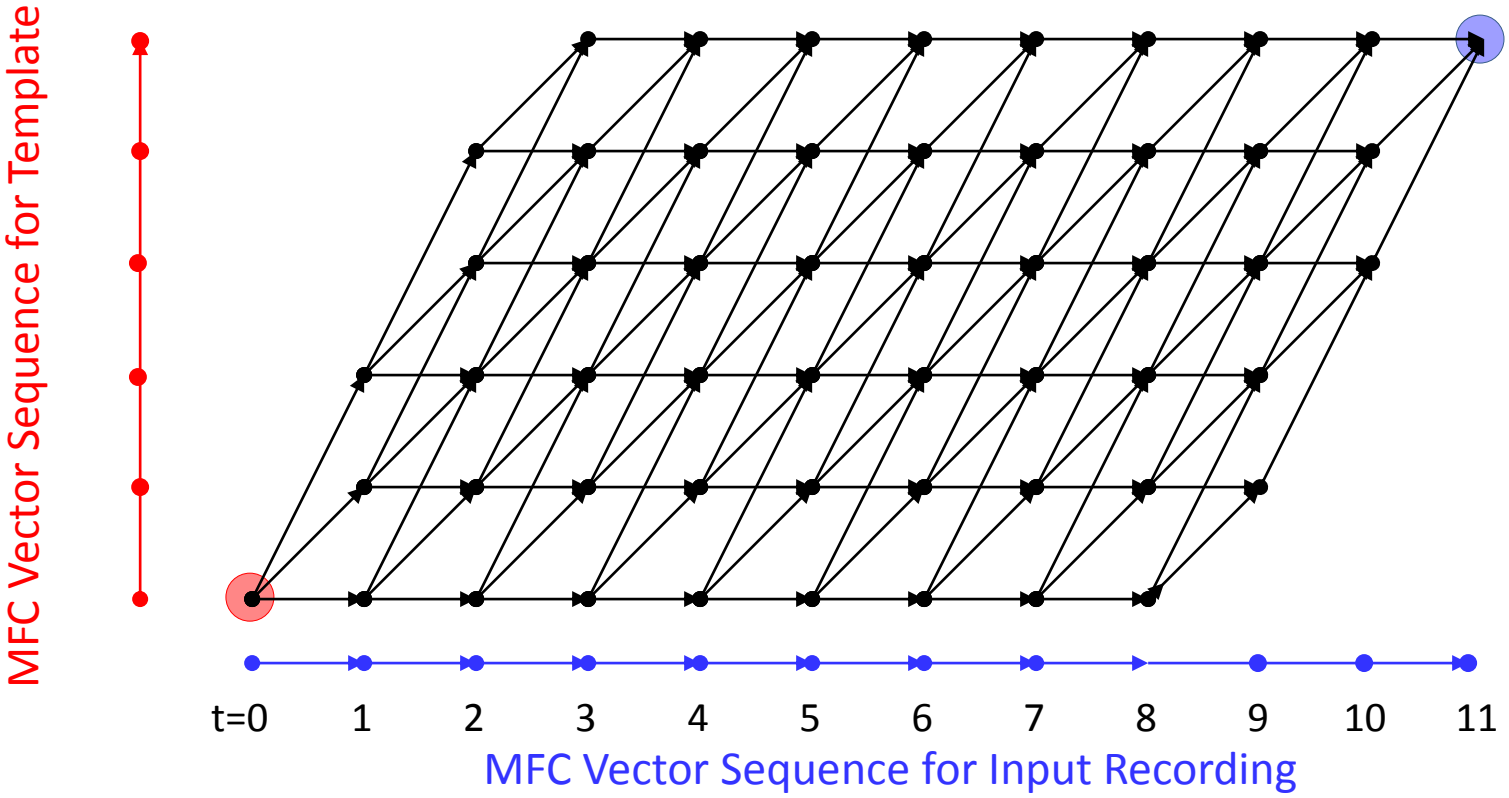
Recap: Isolated word Speech Recognition



- The "compare" operation finds the distance between example (training) recordings, *i.e.* templates of the words and the new input recording

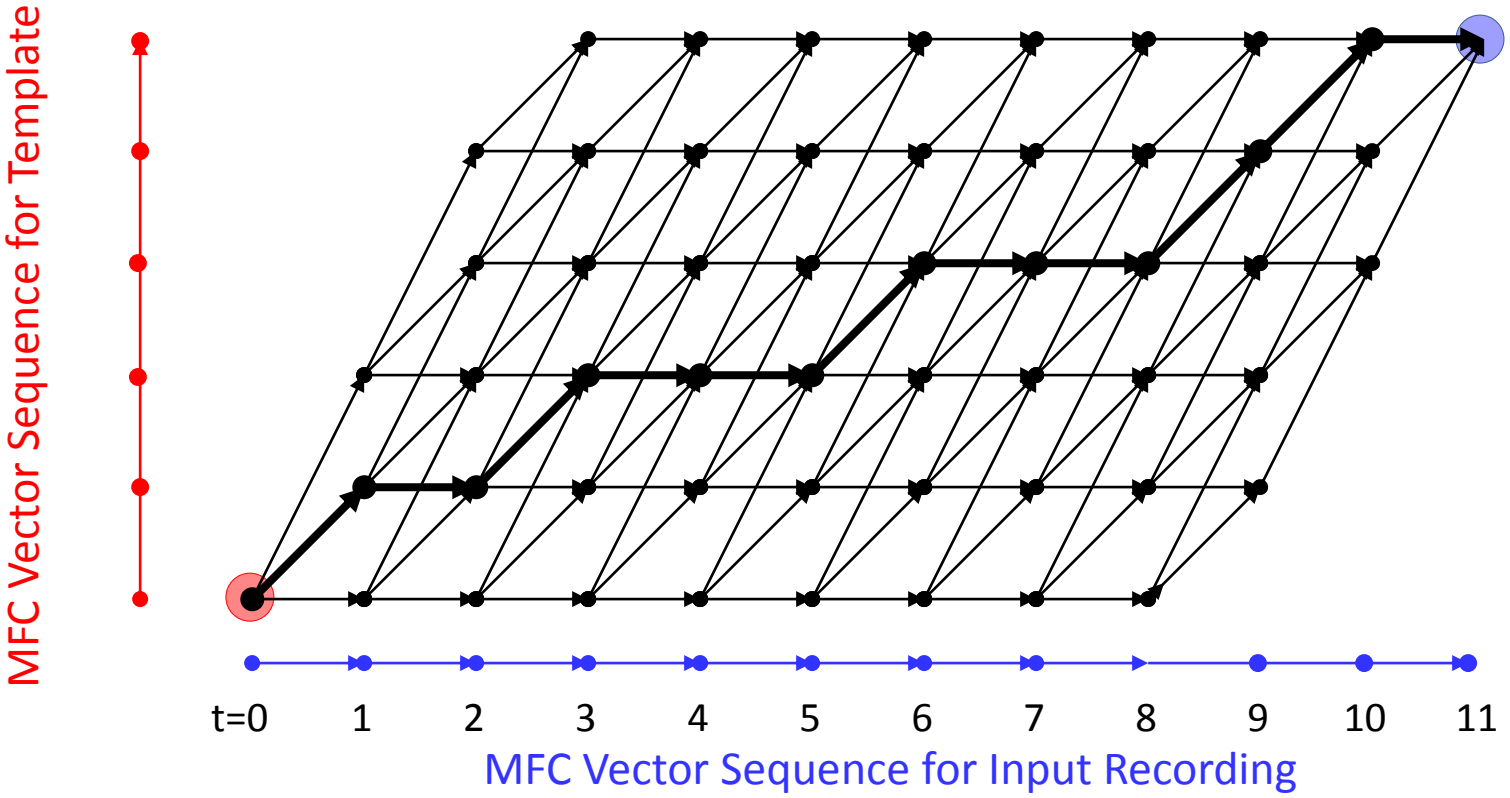
Recap: DTW based comparison of vector sequences

- Trellis for alignment
  - Find lowest-cost path from start node (Red) to sink node (blue)



Recap: DTW based comparison of vector sequences

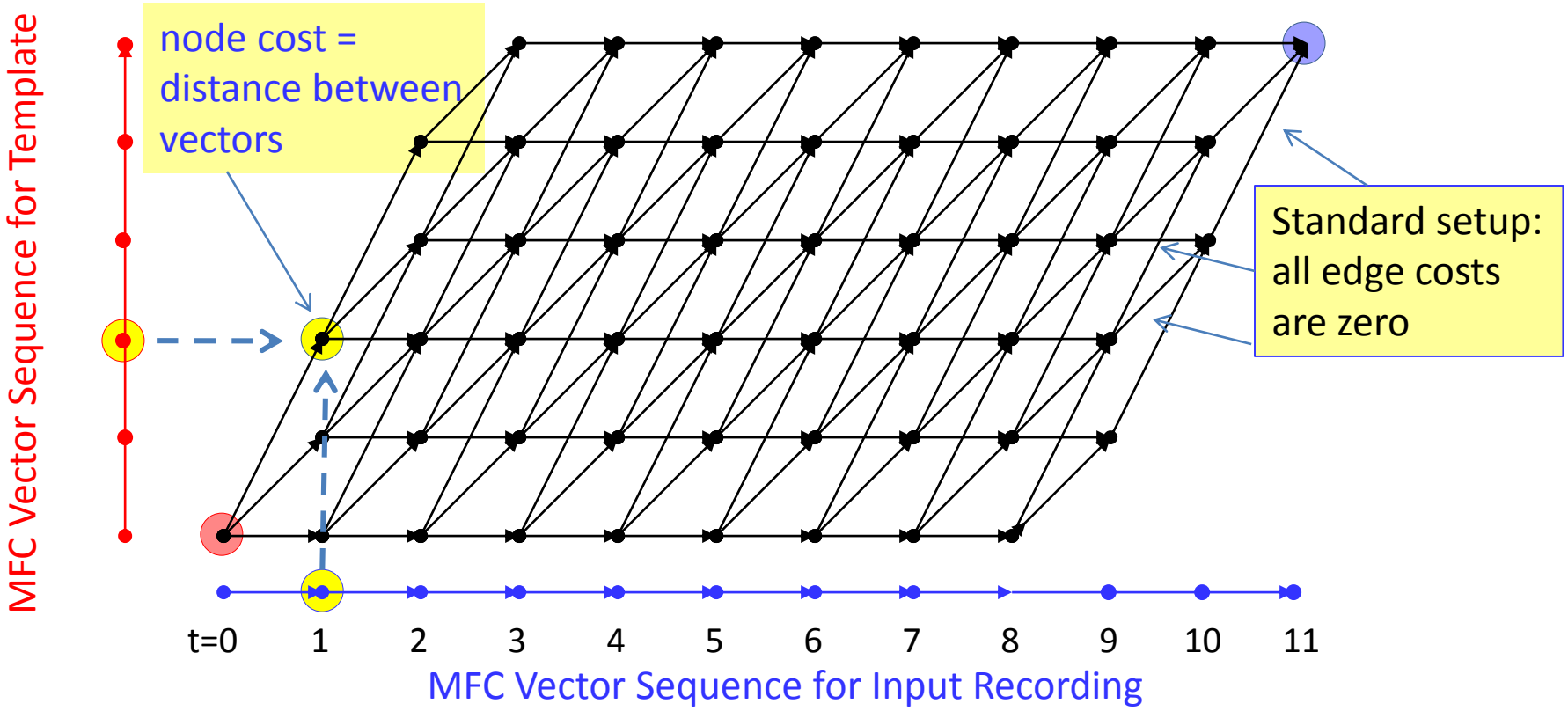
- Trellis for alignment
  - Find lowest-cost path from start node (Red) to sink node (blue)



- Cost of lowest-cost path = distance between template and input
  - The lowest-cost path gives us the alignment between the two sequences

Recap: DTW based comparison of vector sequences

- Trellis for alignment
  - Find lowest-cost path from start node (Red) to sink node (blue)



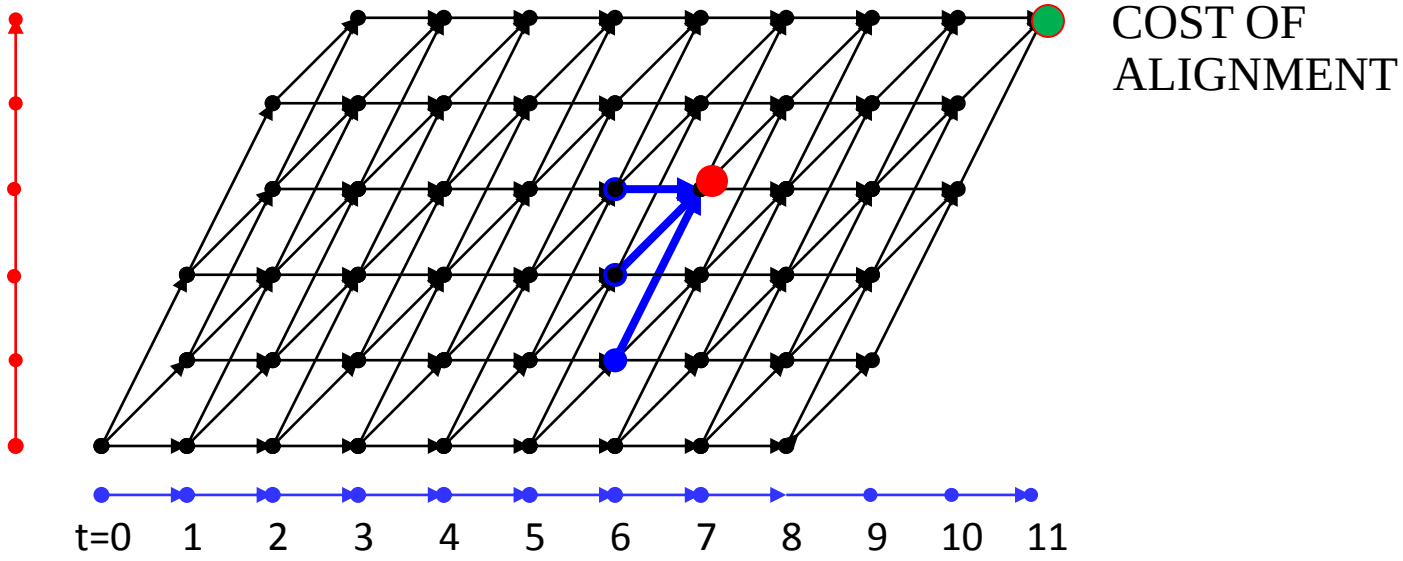
- Cost of lowest-cost path = distance between template and input
  - The lowest-cost path gives us the alignment between the two sequences

DTW: Dynamic Programming Algorithm

- $P_{i,j}$ = best path cost from origin to node $[i,j]$
  - i-th template frame aligns with j-th input frame
- $C_{i,j}$ = local node cost of aligning template frame i to input frame j

$$P_{i,j} = \min (P_{i,j-1} + C_{i,j},\ P_{i-1,j-1} + C_{i,j},\ P_{i-2,j-1} + C_{i,j})$$
$$= \min (P_{i,j-1},\ P_{i-1,j-1},\ P_{i-2,j-1}) + C_{i,j}$$

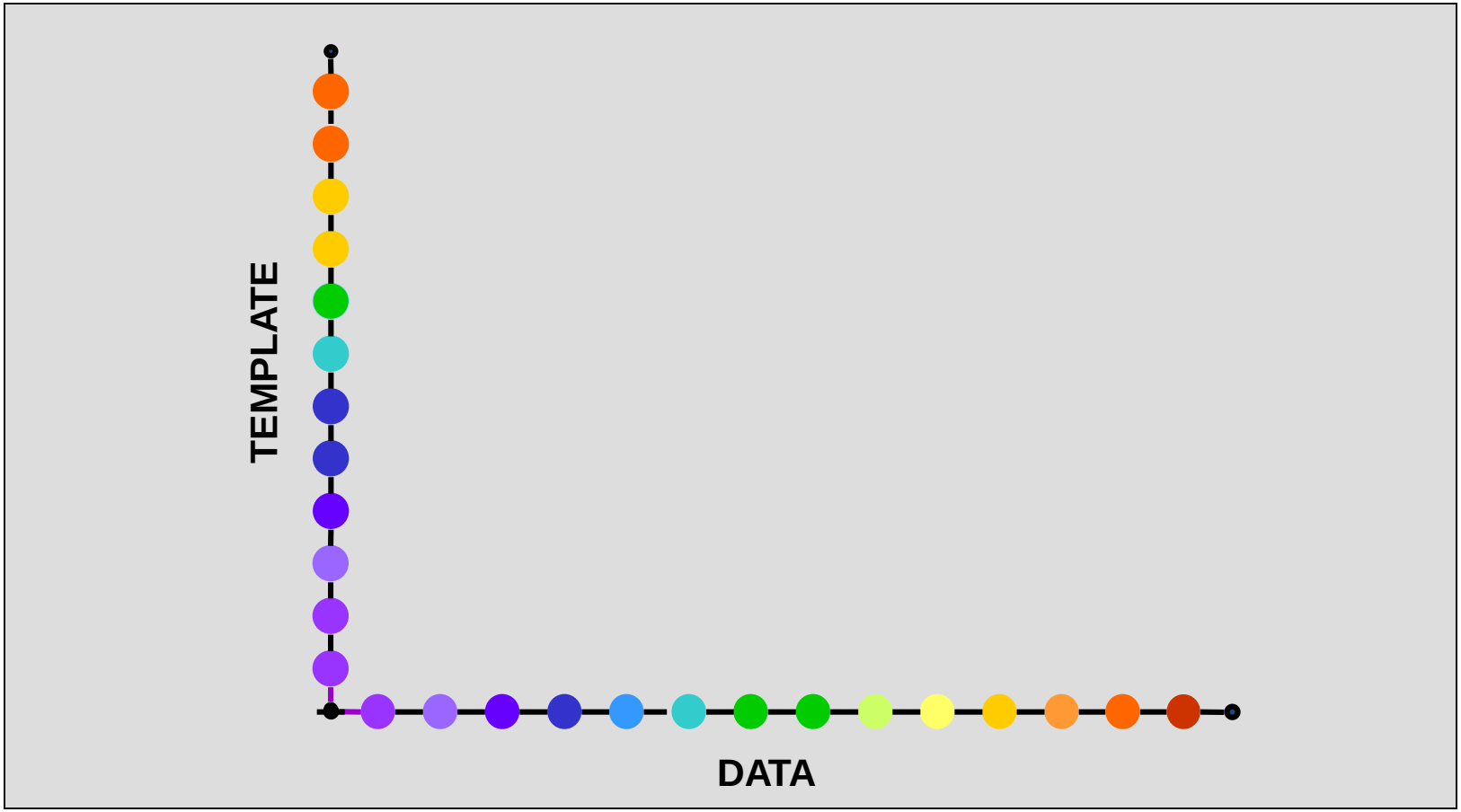
  - Edge costs are 0 in above formulation



Today's Topics

- Generalize DTW based recognition
- Extend to multiple templates
- Move on to Hidden Markov Models
- Look ahead: The fundamental problems of HMMs
  - Introduce the three fundamental problems of HMMs
    - Two of the problems deal with *decoding* using HMMs, solved using the *forward* and *Viterbi* algorithms
    - The third dealing with estimating HMM parameters (seen later)
  - Incorporating *prior knowledge* into the HMM framework
  - Different types of probabilistic models for HMMs
    - Discrete probability distributions
    - Continuous, mixture Gaussian distributions

DTW Using A Single Template



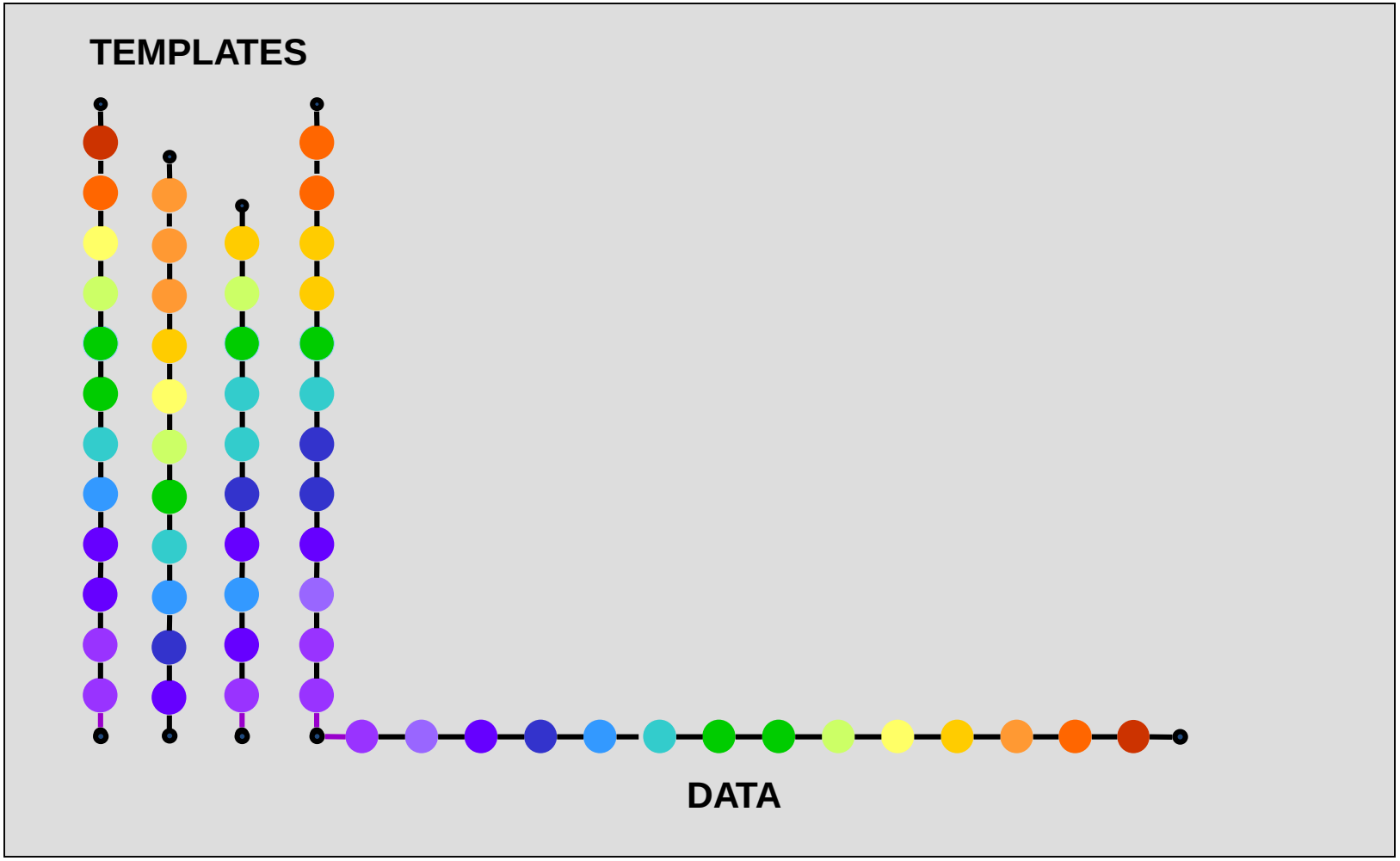
We've seen the DTW alignment of data to model

Limitations of A Single Template

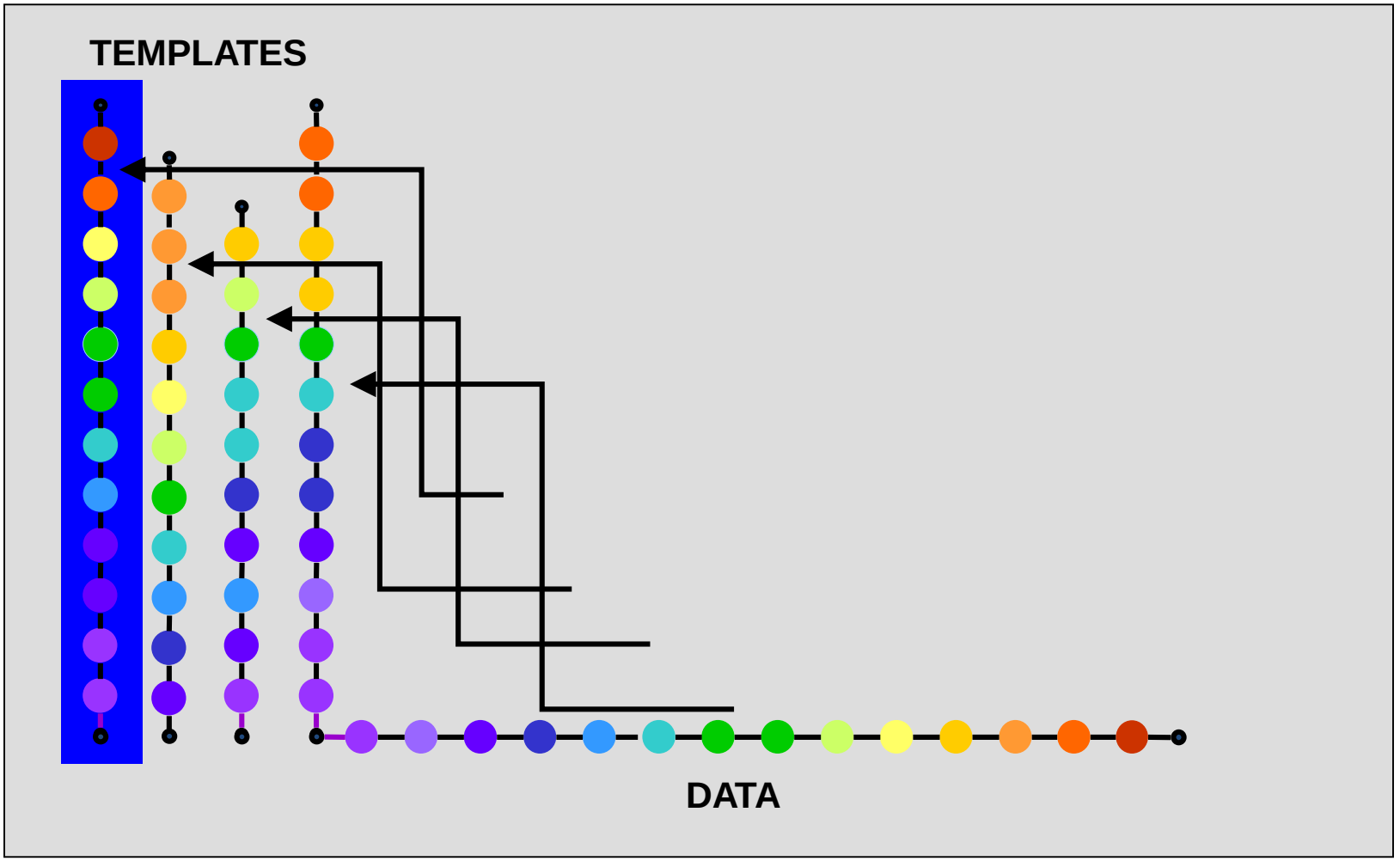
- A single template cannot capture all the possible variations in how a word can be spoken

- Alternative: use multiple templates for each word
  - Match the input against each one

DTW with multiple templates



DTW with multiple templates



Each template warps differently to best match the input; the best matching template is selected

Problem With Multiple Templates

- Finding the best match requires the evaluation of many more templates (depending on the number)
  - This can be computationally expensive
    - Important for handheld devices, even for small-vocabulary applications
    - Think battery life!
  - Need a method for reducing multiple templates into a single one
- Even multiple templates do not cover the *space* of possible variations
  - Need mechanism of generalizing from the templates to include data not seen before
- We can achieve both objectives by *averaging* all the templates for a given word

Generalizing from Templates

- Generalization implies going from the given templates to one that also represents others that we have not seen
- Taking the average of all available templates may represent the recorded templates less accurately, but will represent other unseen templates more robustly
- A general template (for a word) should capture all salient characteristics of the word, and no more
  - Goal: Improving accuracy
- We will consider several steps to accomplish this

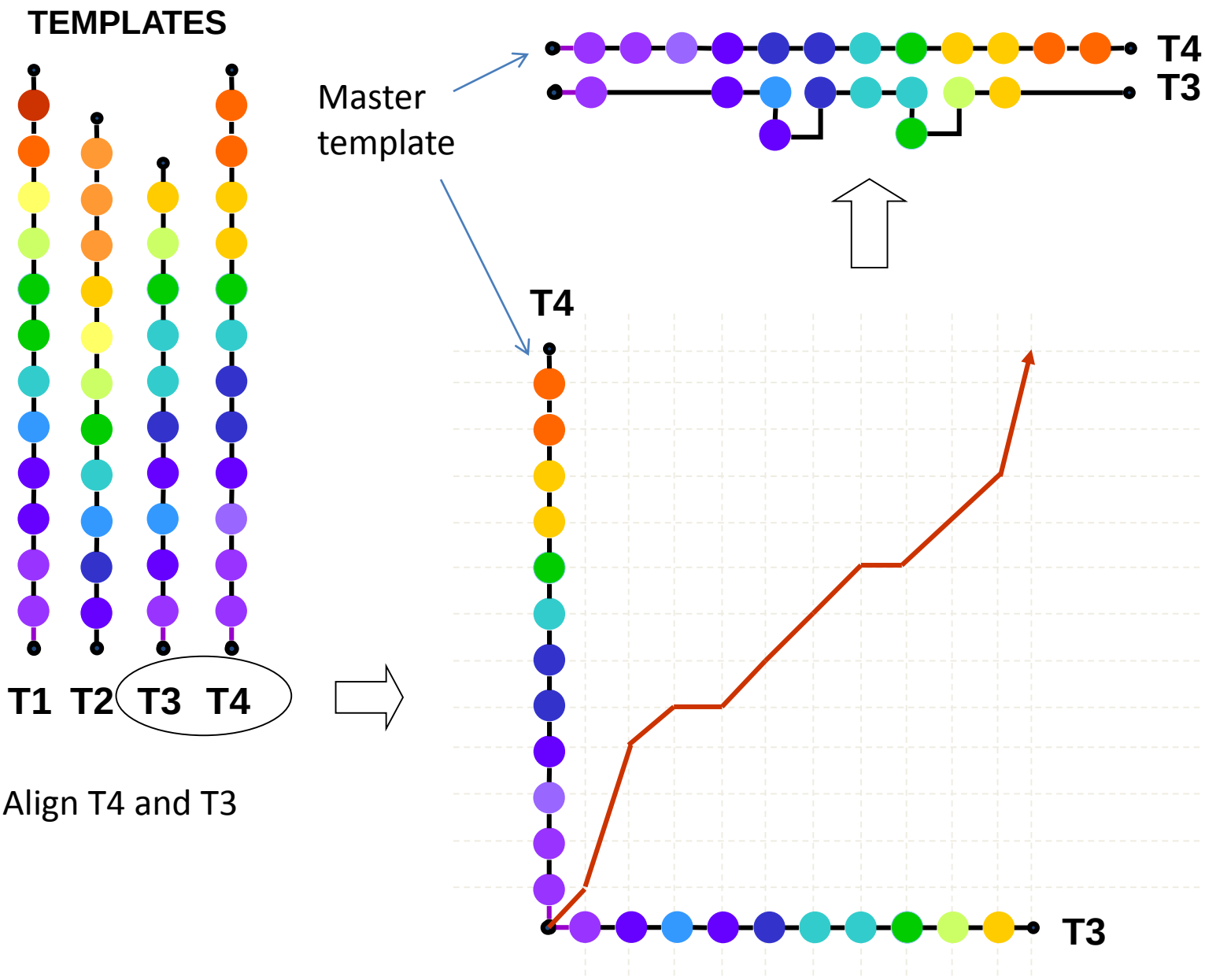
Improving the Templates

- Generalization by averaging the templates
- Generalization by reducing template length
- Accounting for variation within templates represented by the reduced model
- Accounting for varying segment lengths

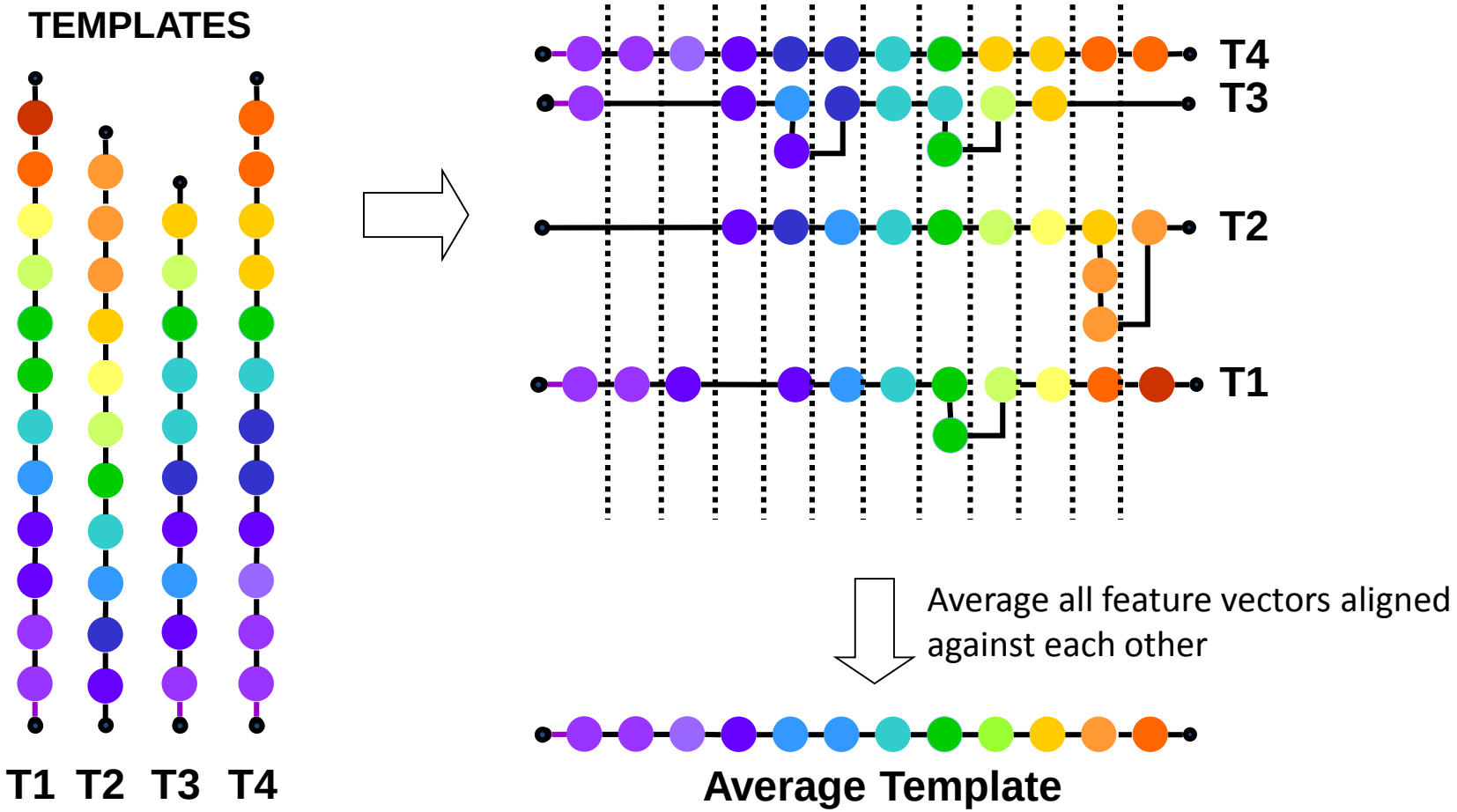
Template Averaging

- How can we average the templates when they're of different lengths?
  - Somehow need to normalize them to each other
- *Solution*: Apply DTW (of course!)
  - Pick one template as a "master"
  - Align all other templates to it
  - Use the alignments generated to compute their average
- *Note*: Choosing a different *master* template will lead to a different average template
  - Which template to choose as the master?
    - Trial and error

DTW with multiple templates



DTW with multiple templates



Align T4/T2 and T4/T1, similarly; then average all of them

Benefits of Template Averaging

- We have eliminated the computational cost of having multiple templates for each word
- Using the averages of the aligned feature vectors *generalizes* from the samples
  - The average is representative of the templates
  - More generally, assumed to be representative of future utterances of the word
- The more the number of templates averaged, the better the generalization

Improving the Templates

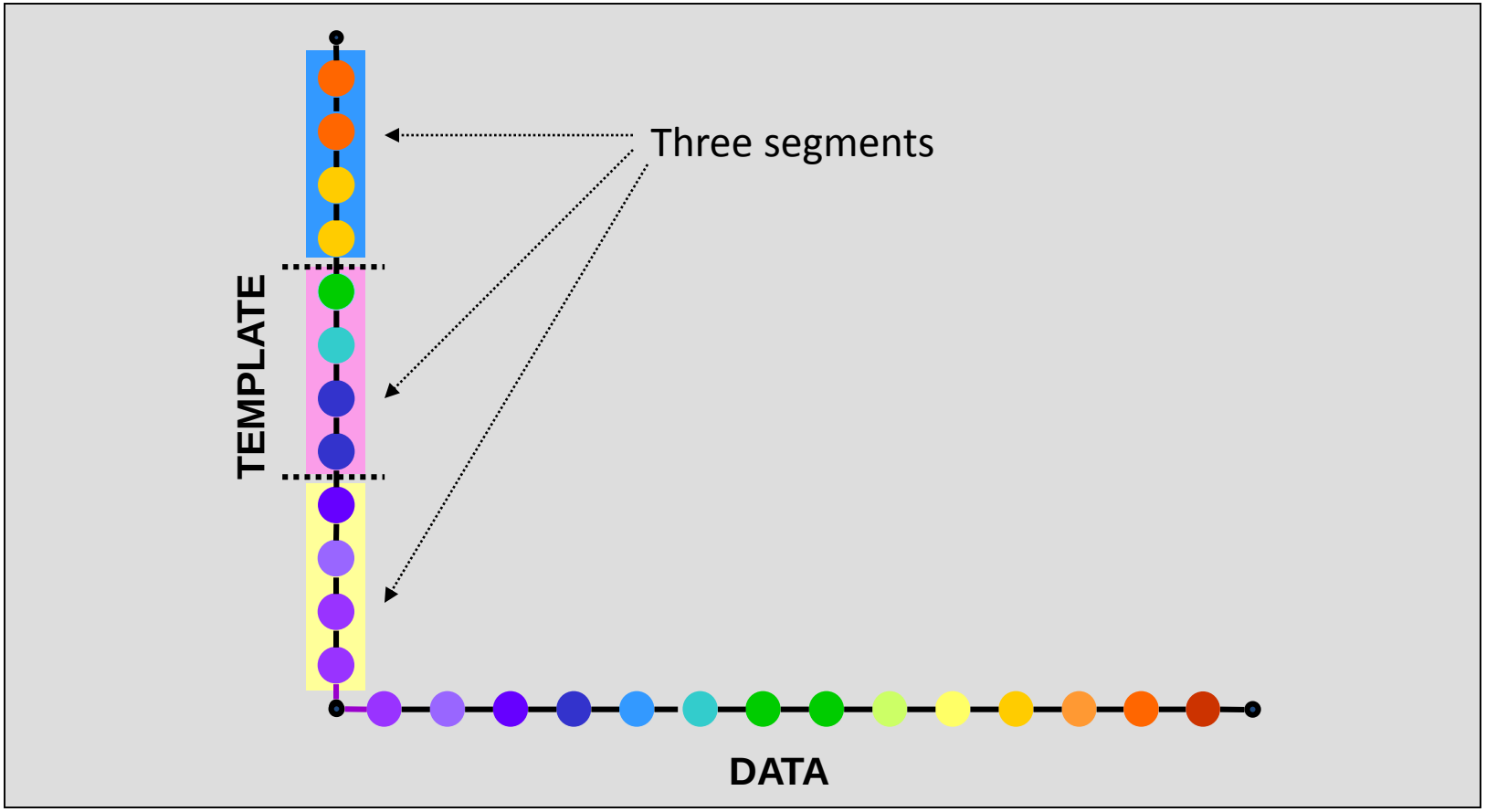
- Generalization by averaging the templates
- Generalization by reducing template length
- Accounting for variation within templates represented by the reduced model
- Accounting for varying segment lengths

Template Size Reduction

- Can we do better? Consider the template for "something":

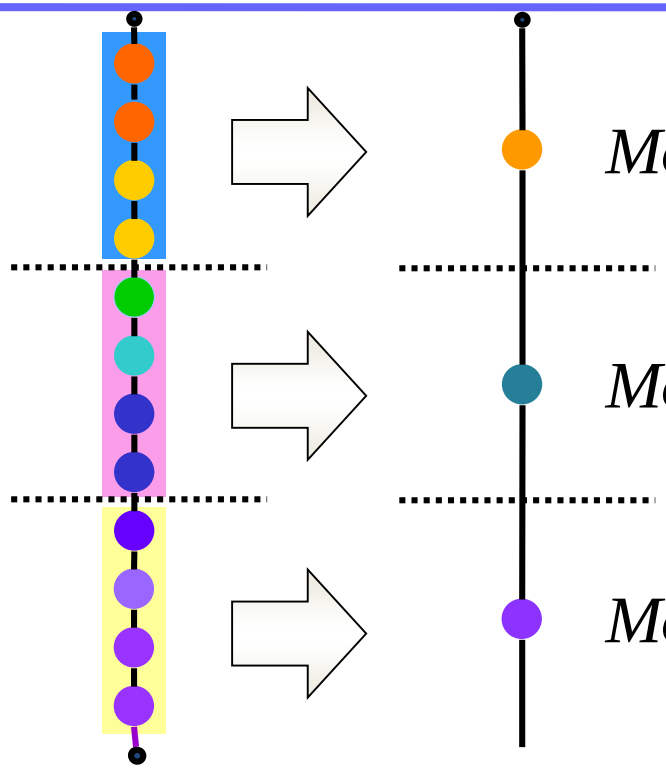
- Here, the template has been manually *segmented* into 6 segments, where each segment is a single phoneme
- Hence, the frames of speech that make up any single segment ought to be fairly alike
- If so, why not replace each segment by a *single* representative feature vector?
  - How? Again by averaging the frames within the segment
- This gives a reduction in the template *size* (memory size)

Example: Single Templates With Three Segments



The feature vectors within each segment are assumed to be similar to each other

Averaging Each Template Segment



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

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

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

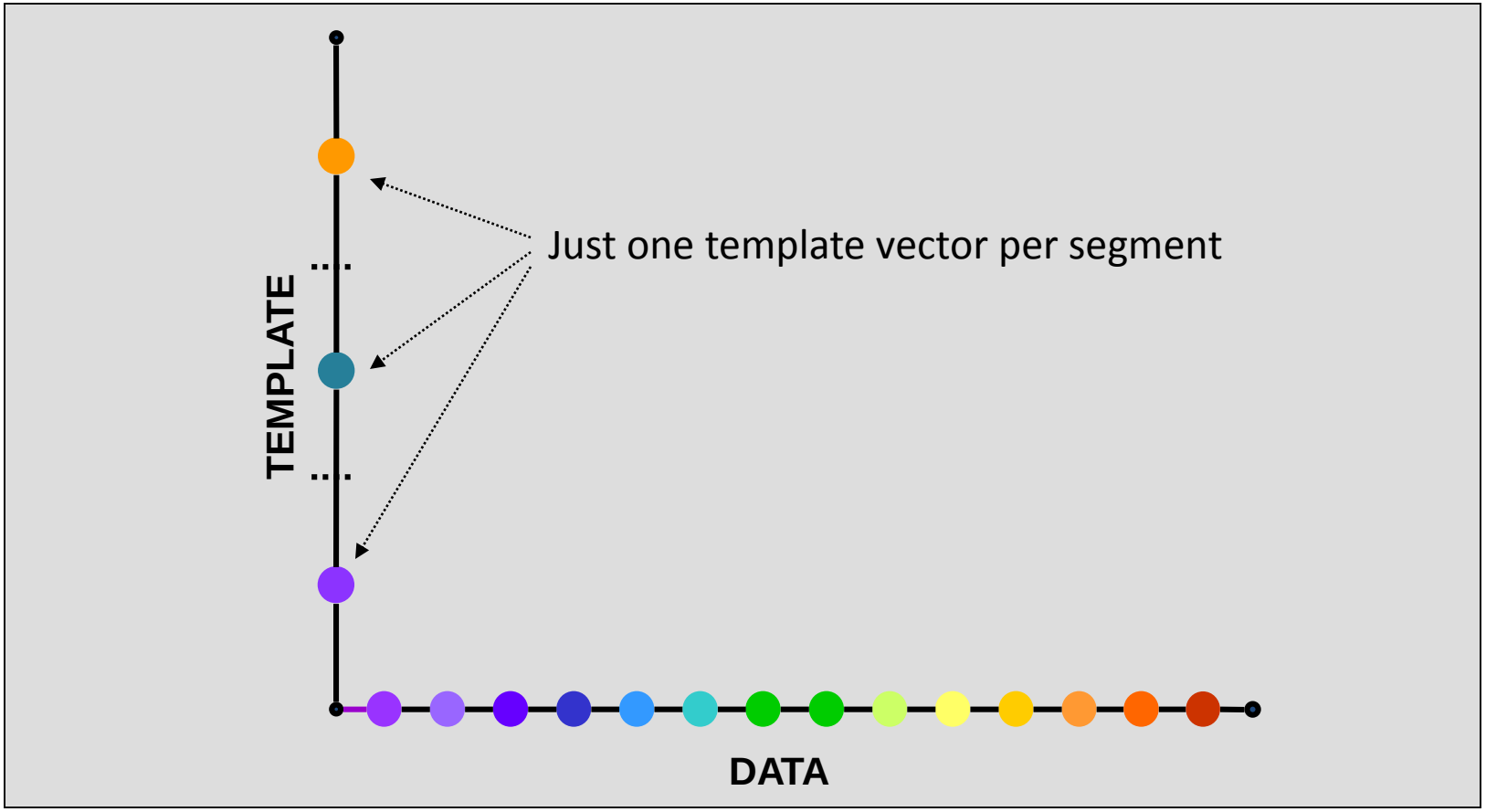
$$m_j = \frac{1}{N_j} \sum_{i \in segment(j)} x(i)$$

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

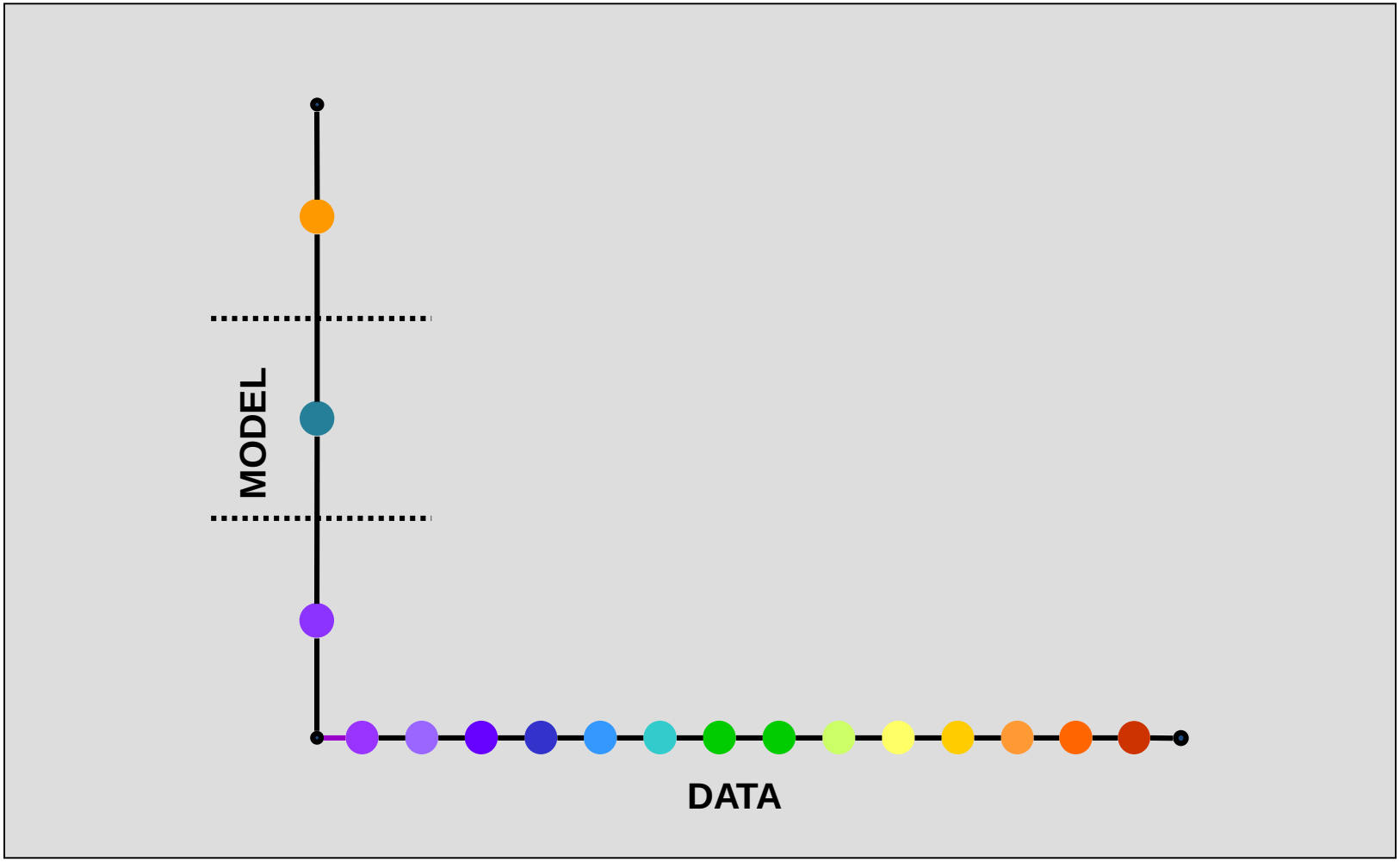
N_j is the number of vectors in the j^{th} segment

$x(i)$ is the i^{th} feature vector

Template With One Model Vector Per Segment

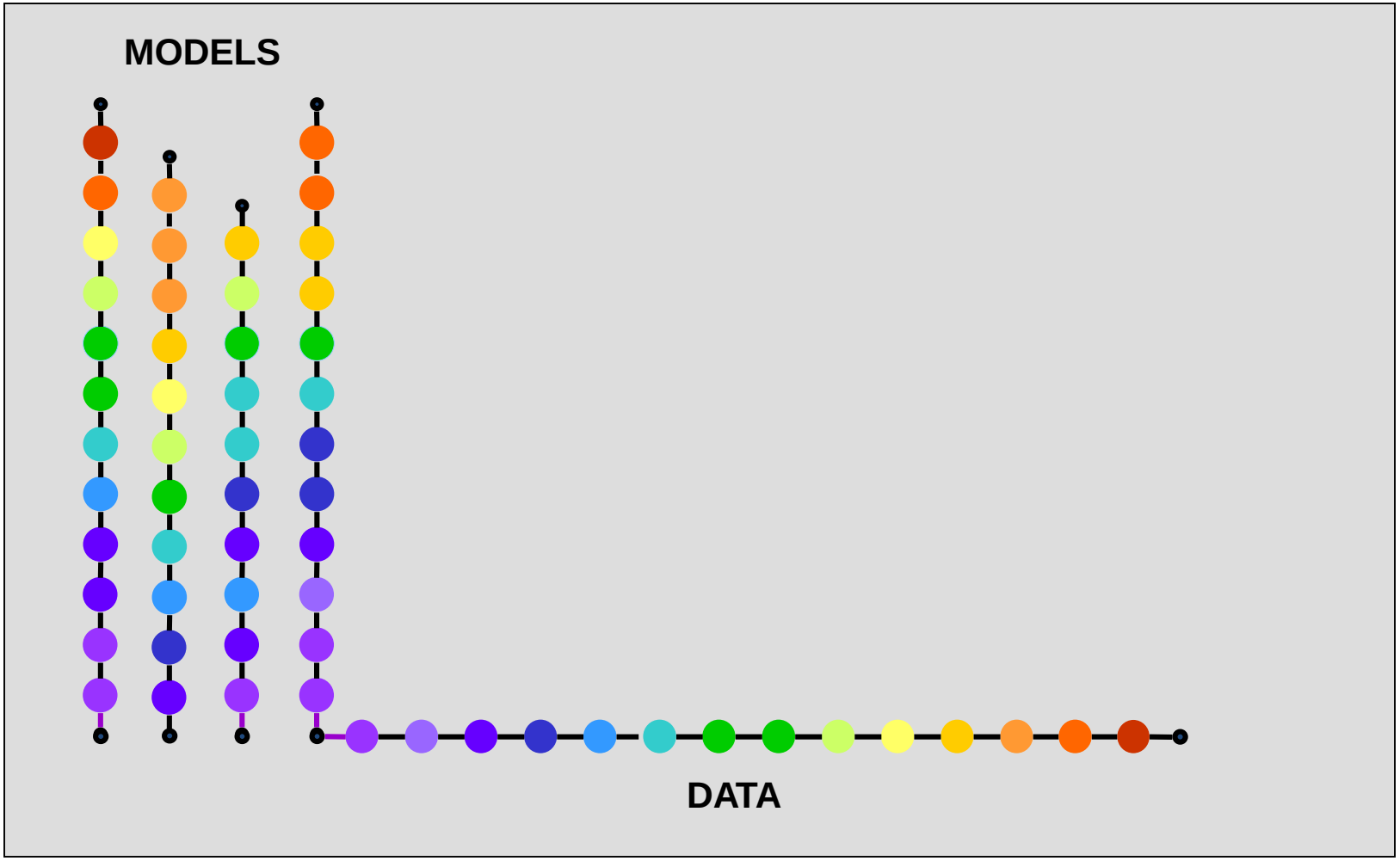


DTW with one model



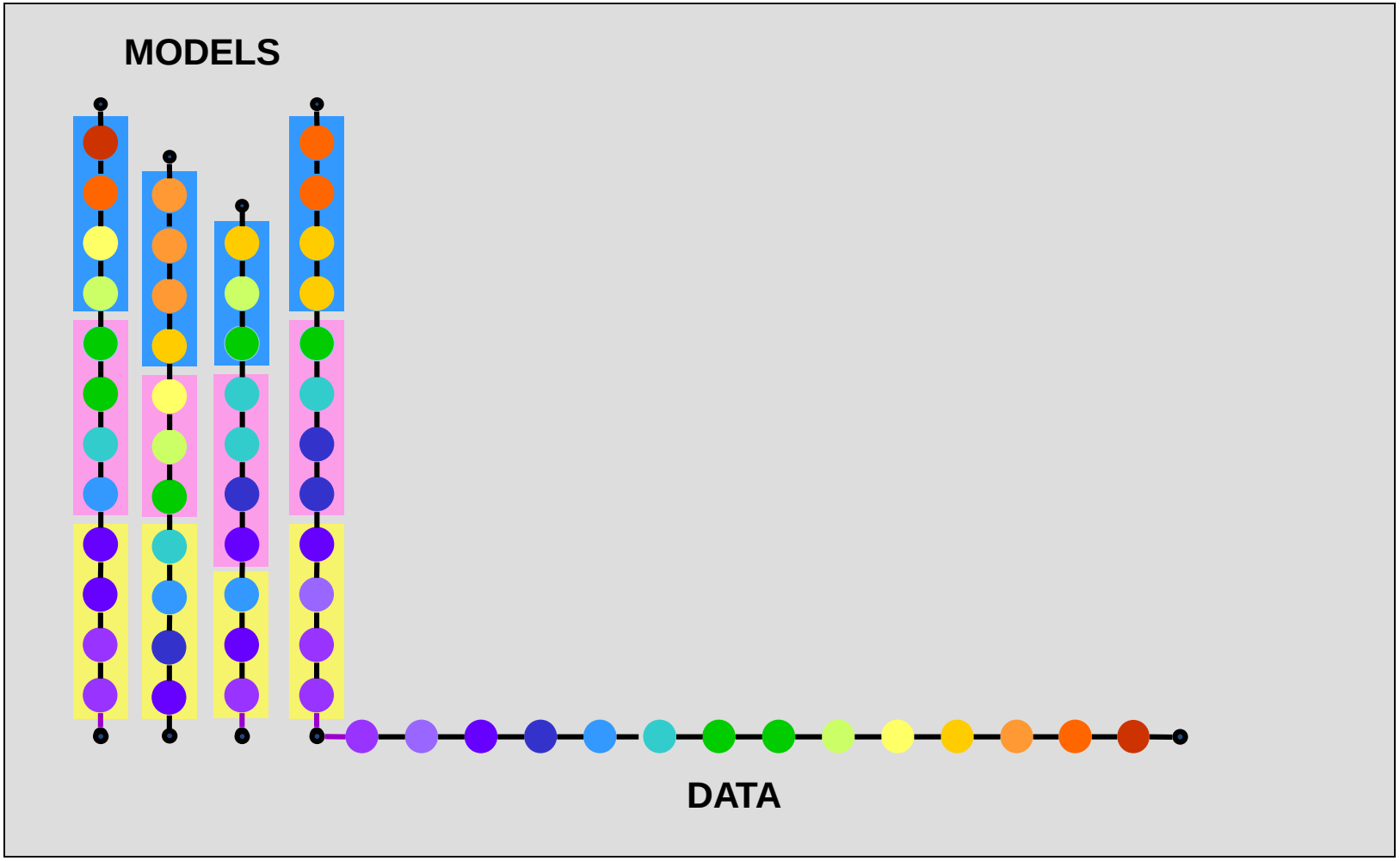
The averaged template is matched against the data string to be recognized
Select the word whose averaed template has the lowest cost of match

DTW with multiple models



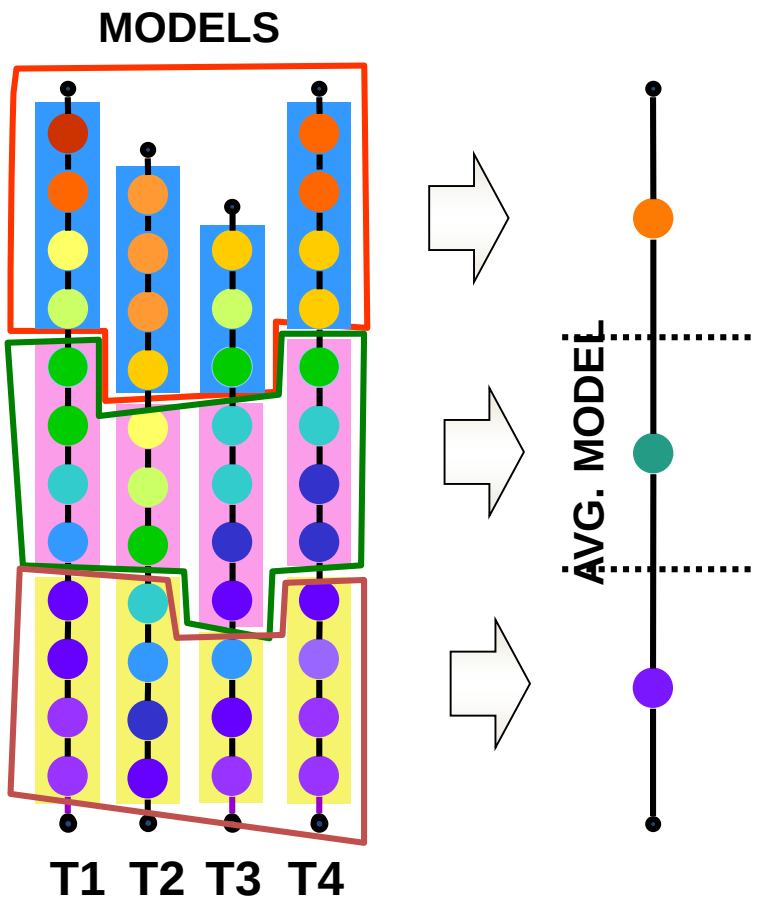
Segment all templates
Average each region into a single point

DTW with multiple models



Segment all templates
Average each region into a single point

DTW with multiple models



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

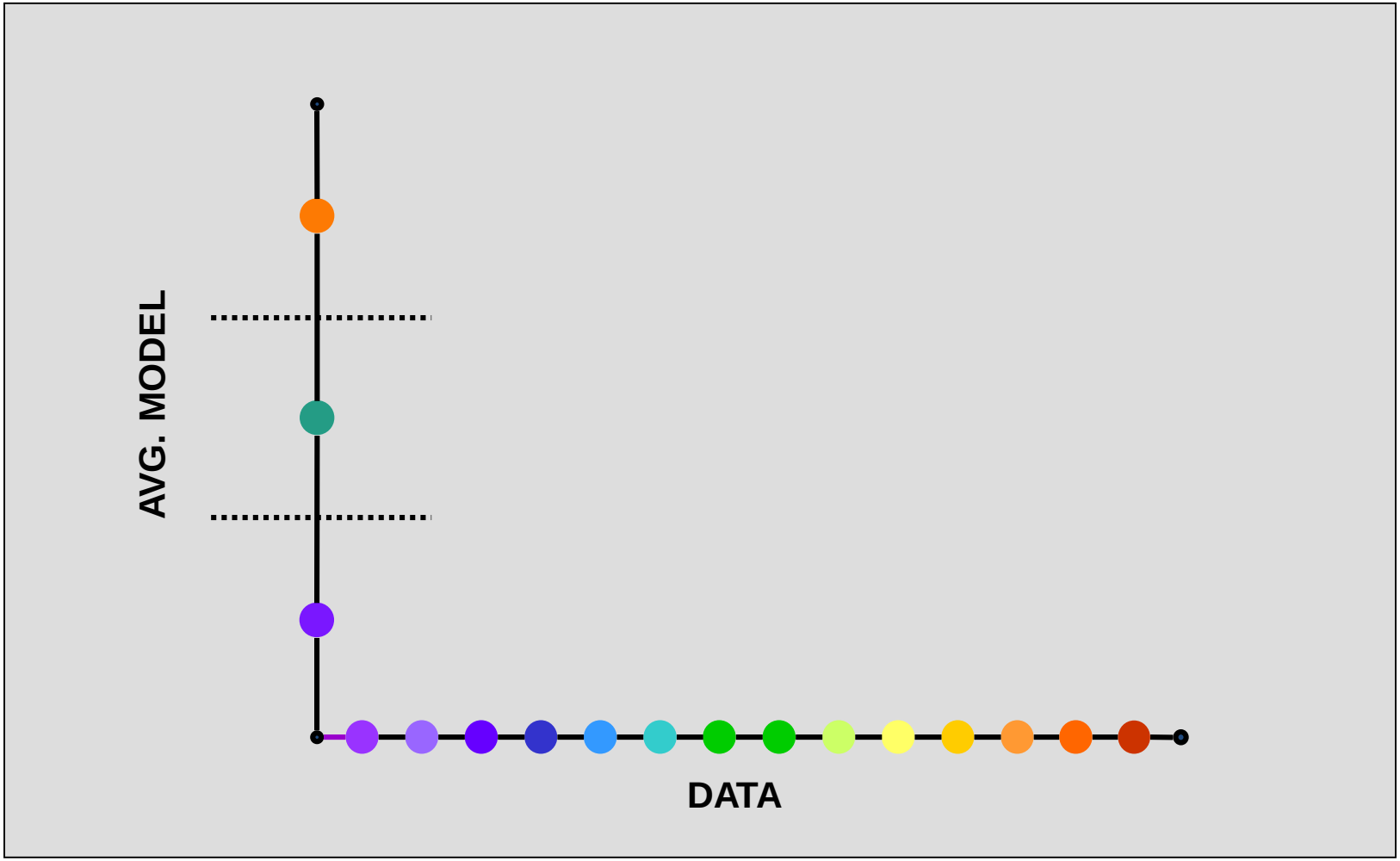
$segment_k(j)$ is the j^{th} segment of the k^{th} training sequence

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

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

DTW with multiple models

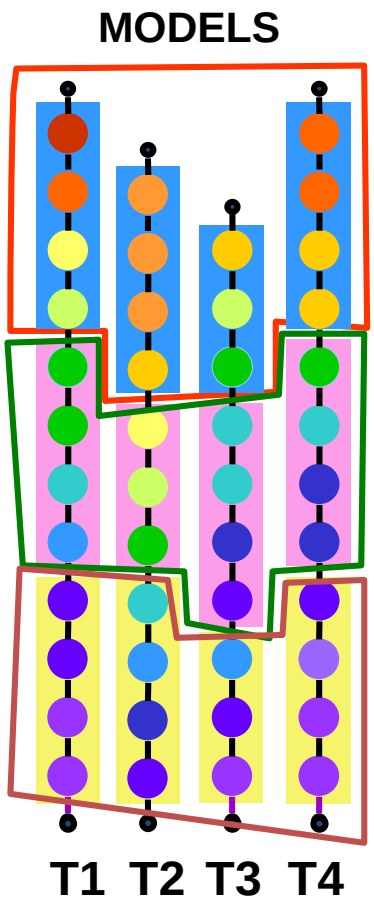


Segment all templates, average each region into a single point
To get a simple average model, which is used for recognition

Improving the Templates

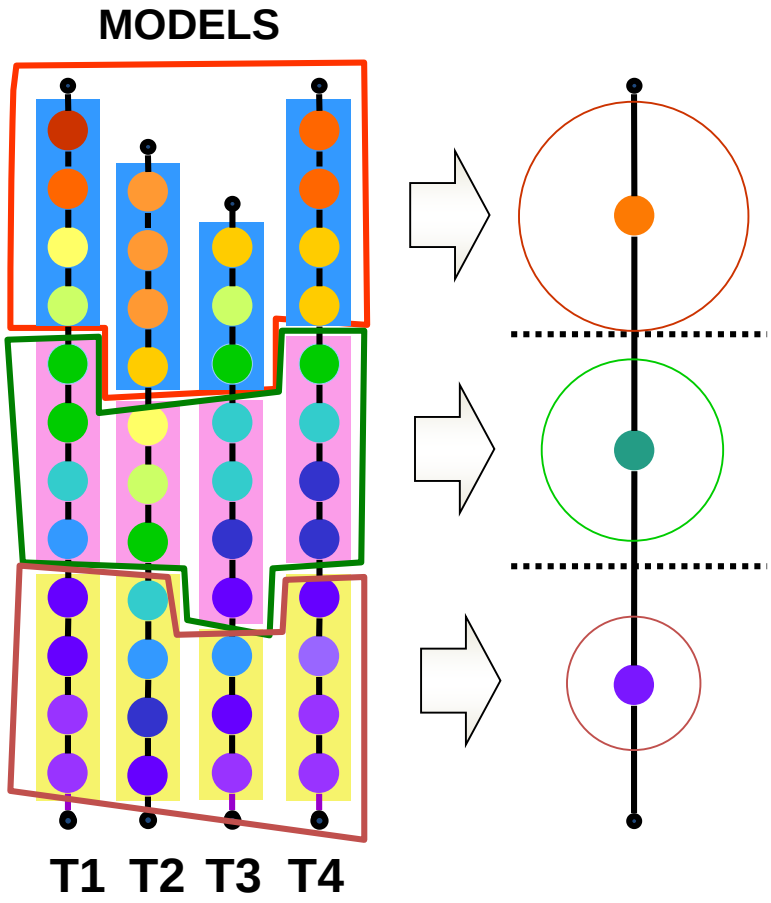
- Generalization by averaging the templates
- Generalization by reducing template length
- Accounting for variation within templates represented by the reduced model
- Accounting for varying segment lengths

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

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

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

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

  - x 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

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

$$d(x, m_j) = -\log\left(Gaussian(x; m_j, C_j)\right) = 0.5\log\left((2\pi)^D |C_j|\right) + 0.5(x - m_j)^T C_j^{-1} (x - m_j)$$

The Covariance

- The variance that we have computed is a *full covariance matrix*
  - And the distance measure requires a matrix inversion

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

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

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1N} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{N1} & \sigma_{N2} & \cdots & \sigma_N^2 \end{pmatrix}$$

- **In practice we assume that all off-diagonal terms in the matrix are 0**
- This reduces our distance metric to:

$$d(x, m_j) = \sum_l \frac{(x_l - m_{j,l})^2}{\sigma_{j,l}^2}$$

$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_N^2 \end{pmatrix}$$

- Where the individual variance terms σ^2 are

$$\sigma_{j,l}^2 = \frac{1}{\sum_k N_k} \sum_k \sum_{i \in segment_k(j)} \left(x_{k,l}(i) - m_{j,l}\right)^2$$

- If we use a negative log Gaussian instead, the modified score (with the diagonal covariance) is

$$d(x, m_j) = 0.5 \sum_l \log(2\pi\sigma_{j,l}^2) + 0.5 \sum_l \frac{(x_l - m_{j,l})^2}{\sigma_{j,l}^2}$$

Segmental K-means

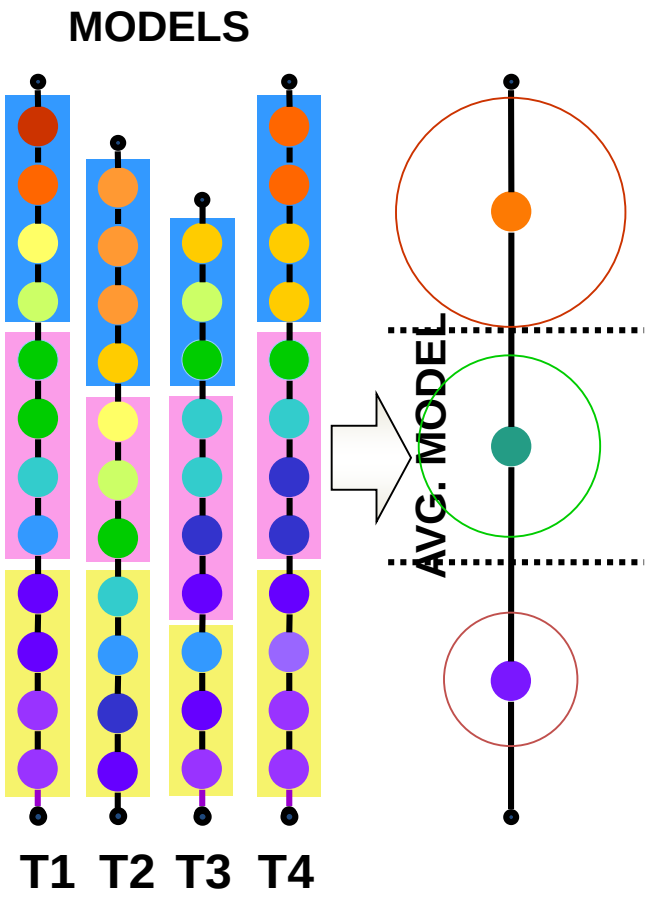
- Simple uniform segmentation of training instances is not the most effective method of grouping vectors in the training sequences

- 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

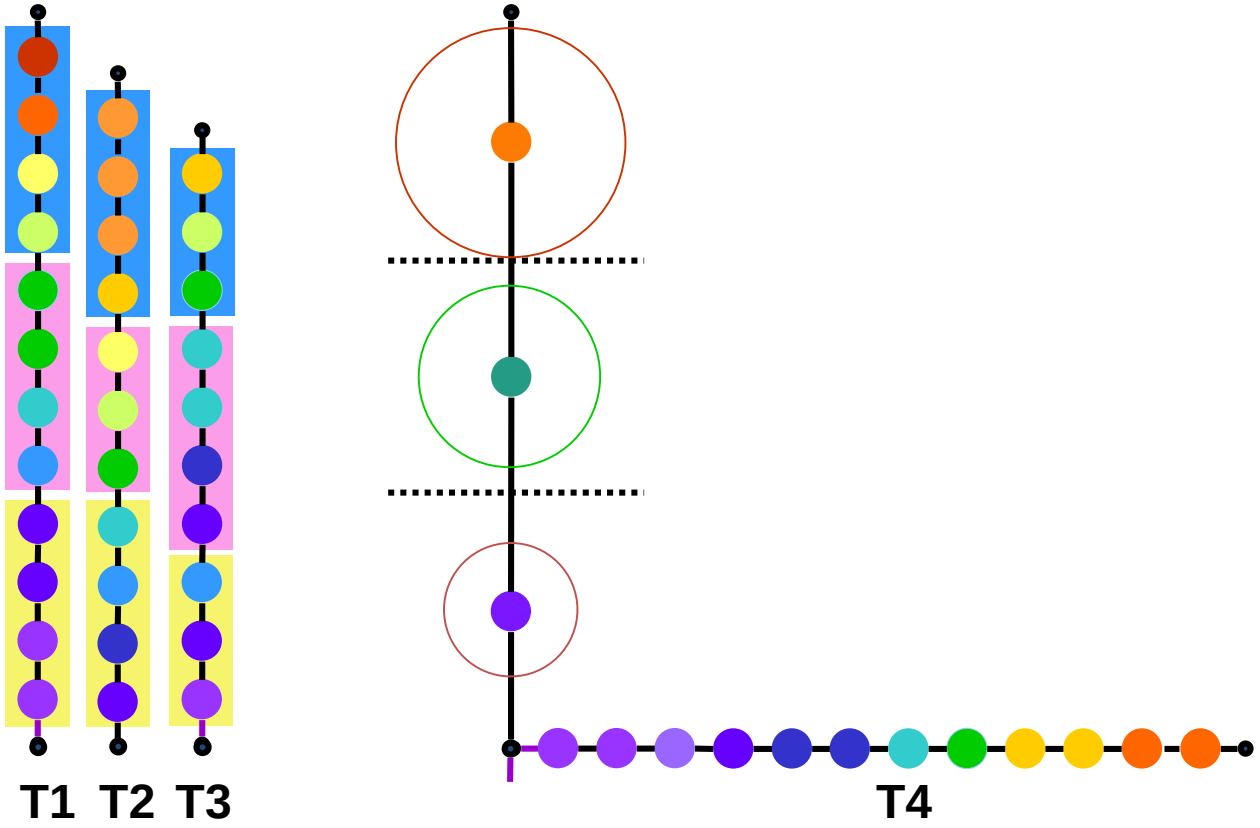
- The segmental K-means procedure is an iterative procedure to estimate the optimal segmentation

Alignment for training a model from multiple vector sequences



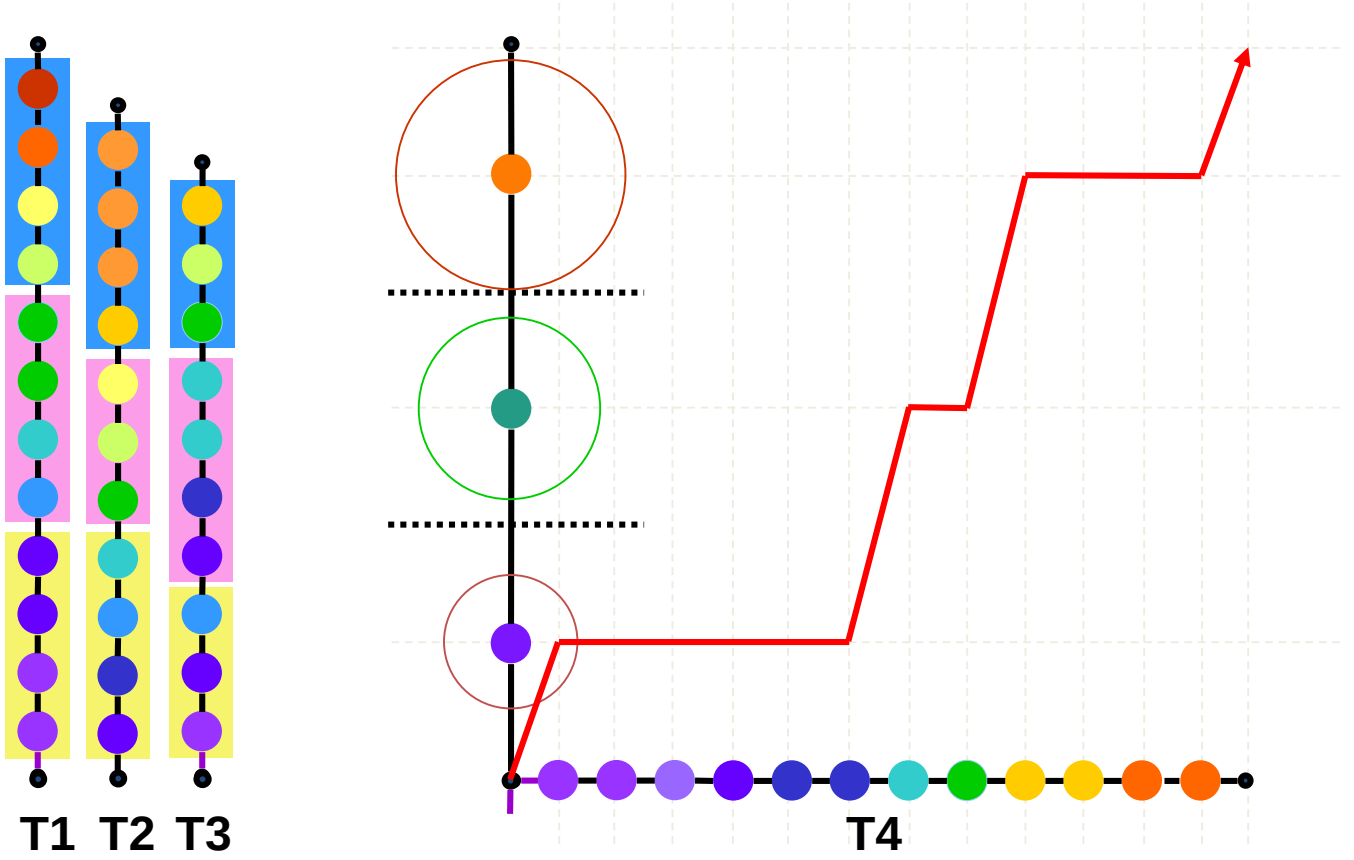
Initialize by uniform segmentation

Alignment for training a model from multiple vector sequences



Initialize by uniform segmentation

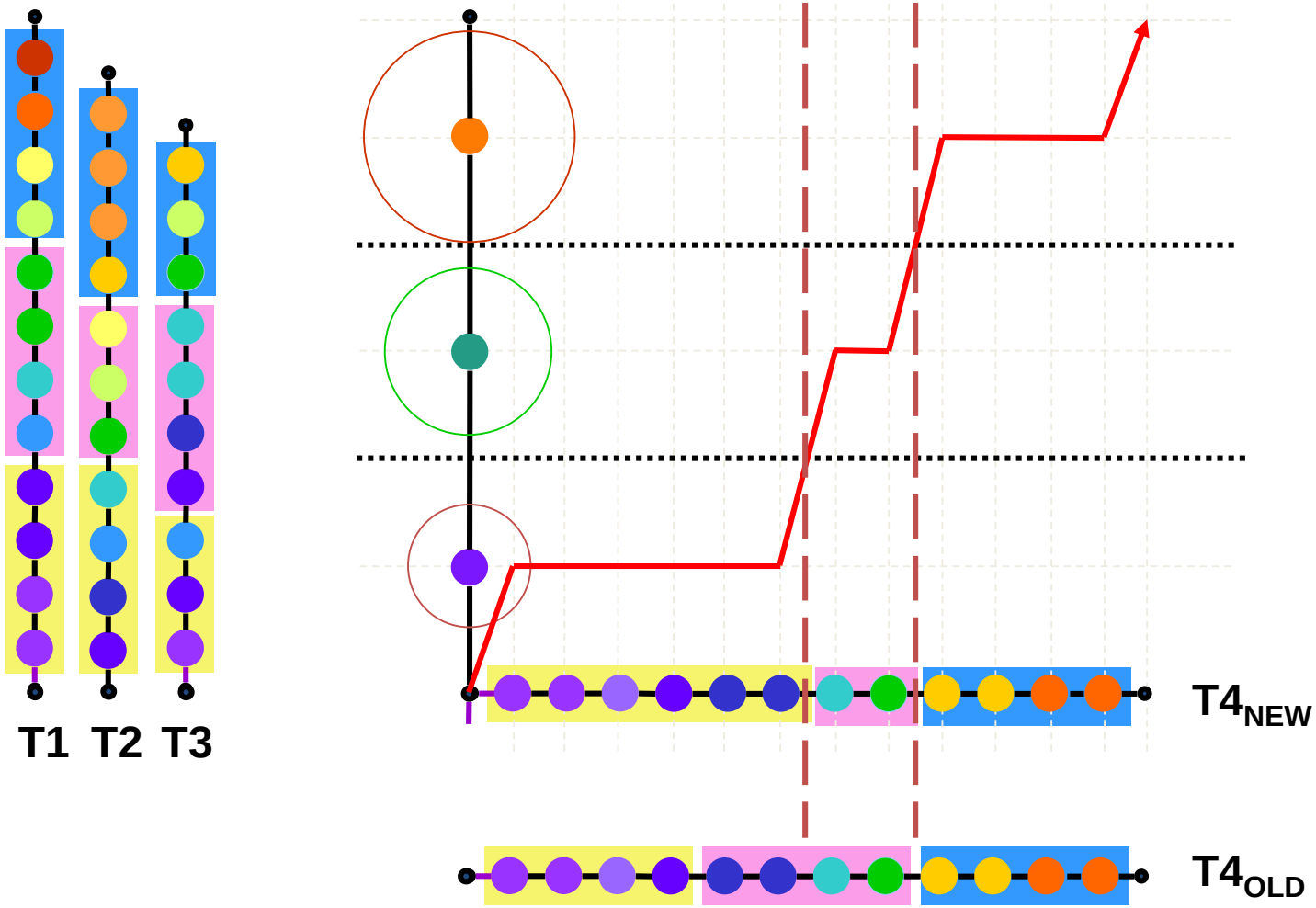
Alignment for training a model from multiple vector sequences



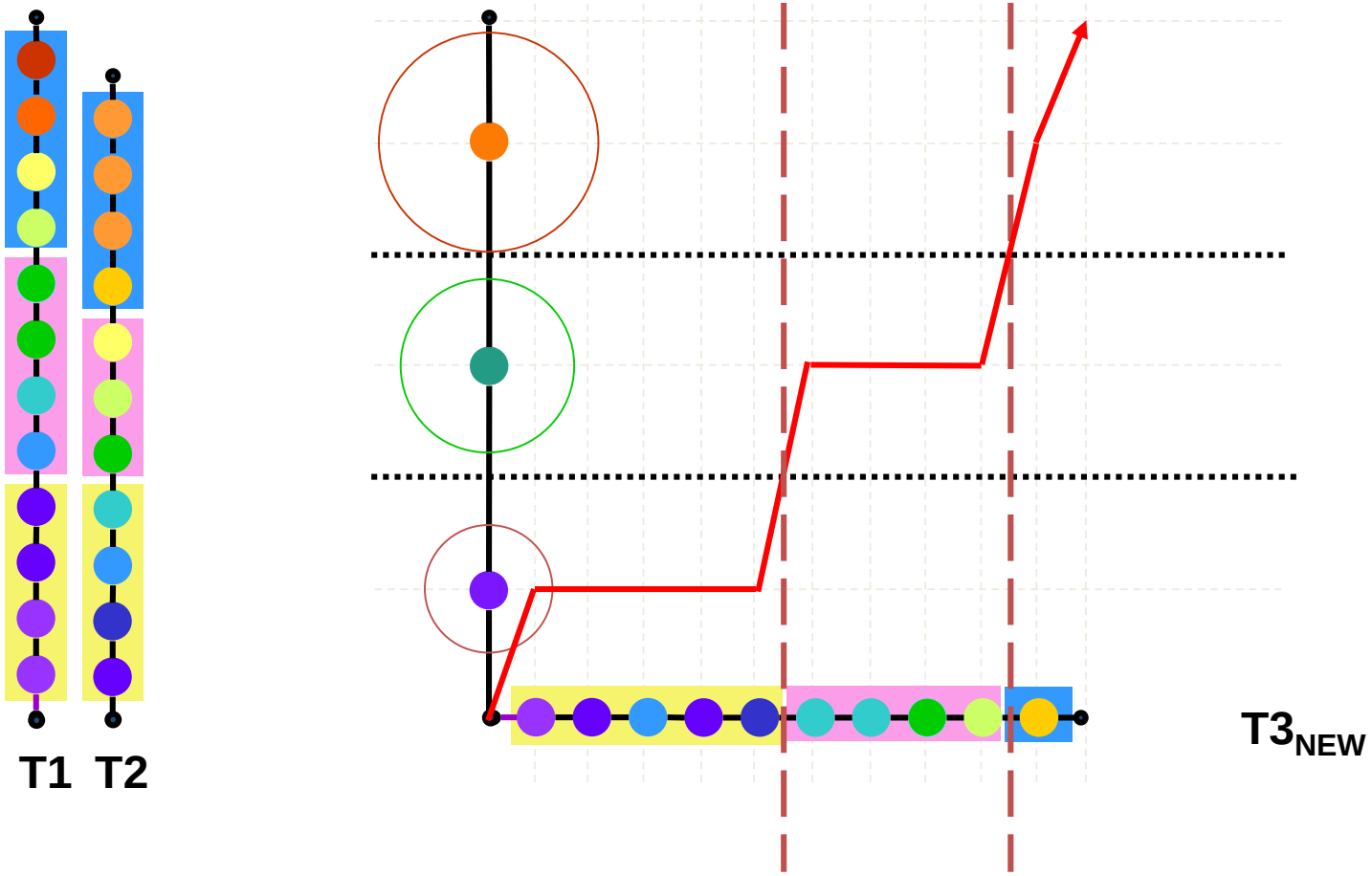
Initialize by uniform segmentation

Align each template to the averaged model to get new segmentations

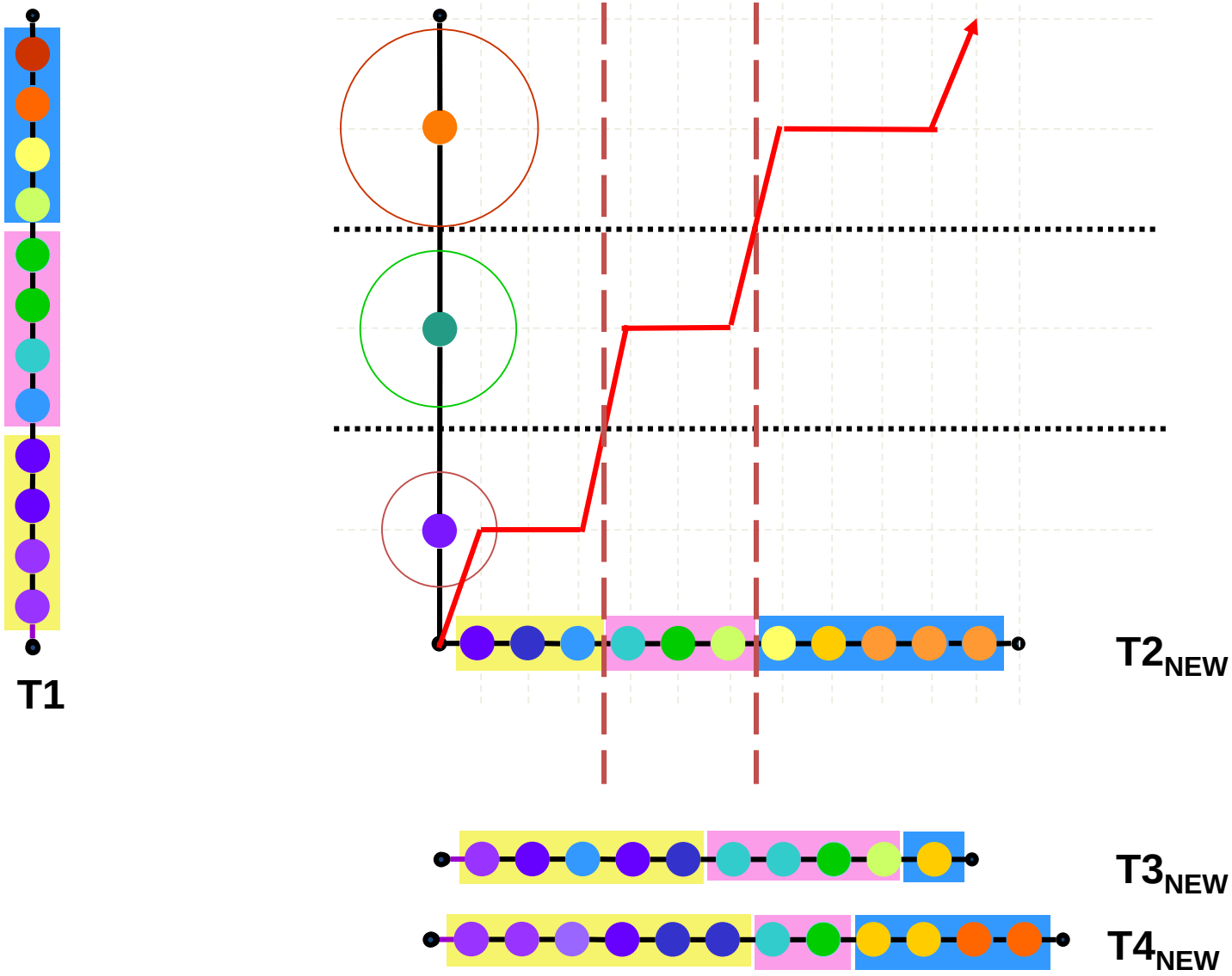
Alignment for training a model from multiple vector sequences



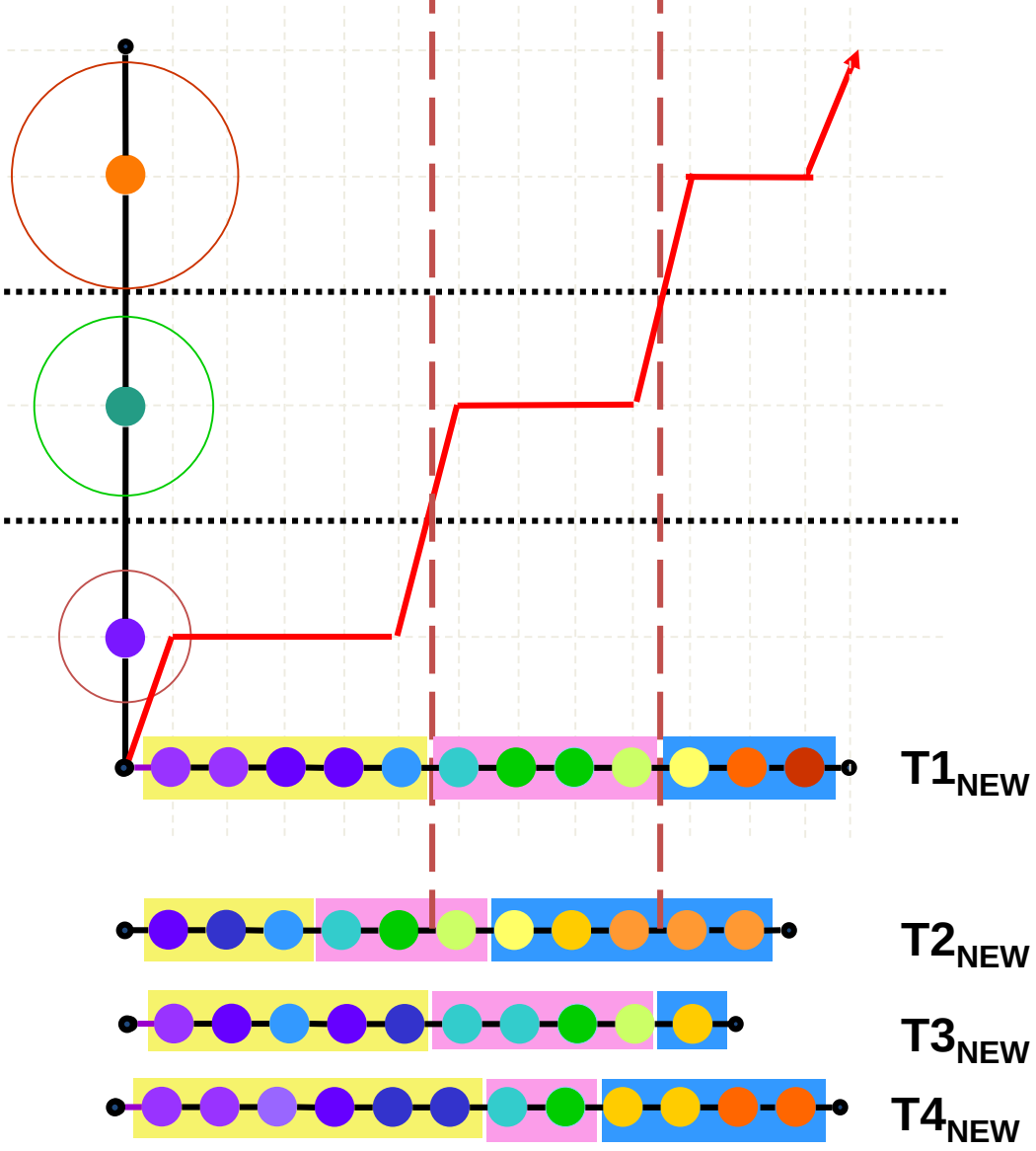
Alignment for training a model from multiple vector sequences



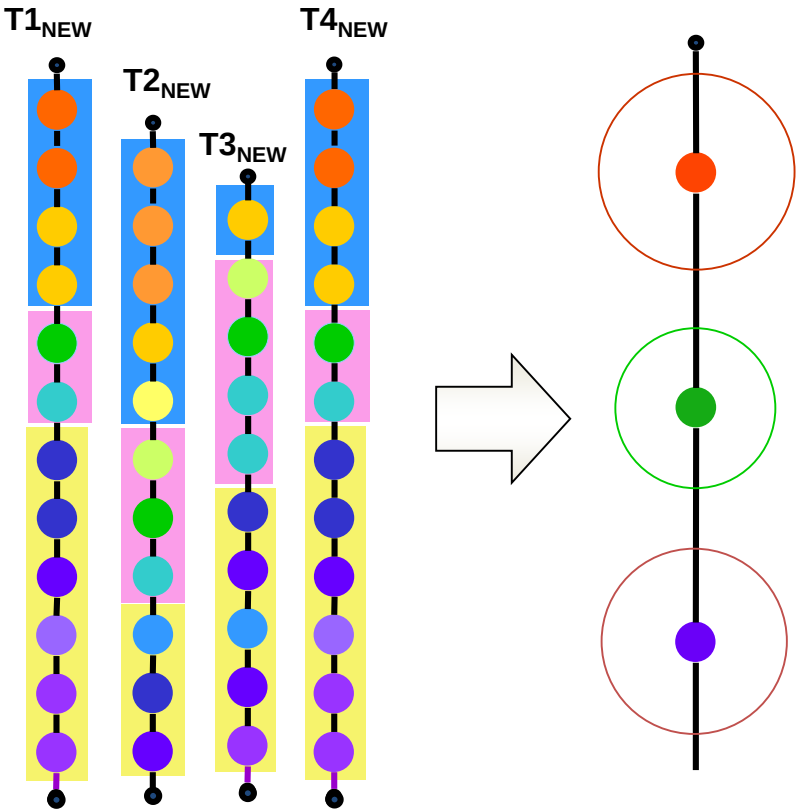
Alignment for training a model from multiple vector sequences



Alignment for training a model from multiple vector sequences

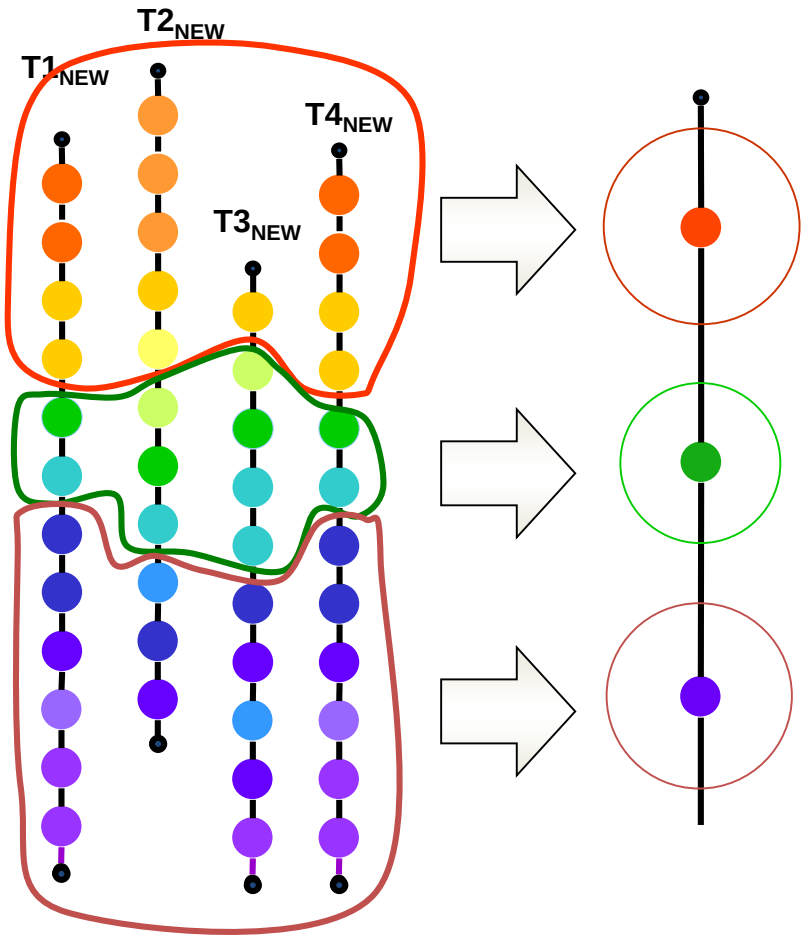


Alignment for training a model from multiple vector sequences

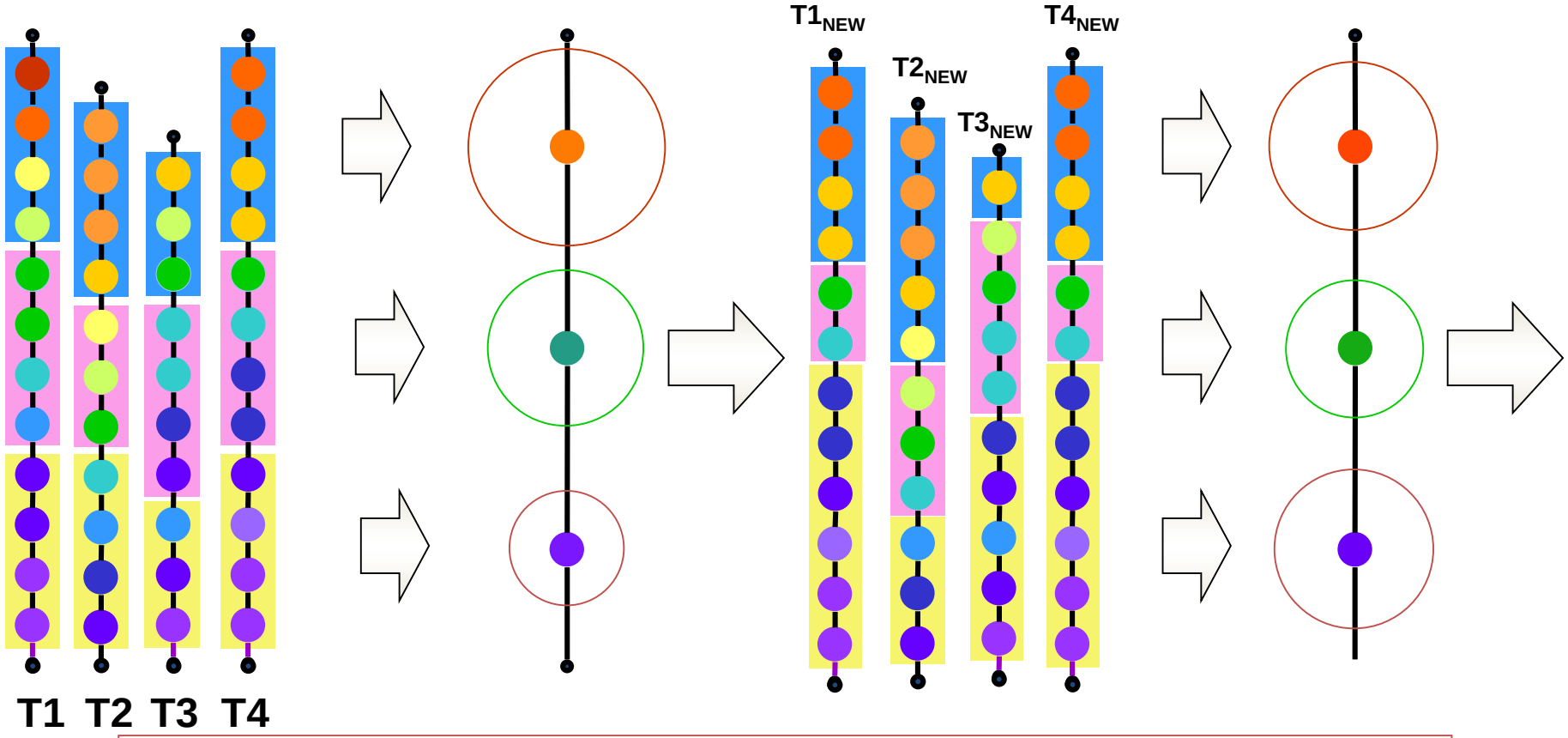


Initialize by uniform segmentation
Align each template to the averaged model to get new segmentations
Recompute the average model from new segmentations

Alignment for training a model from multiple vector sequences



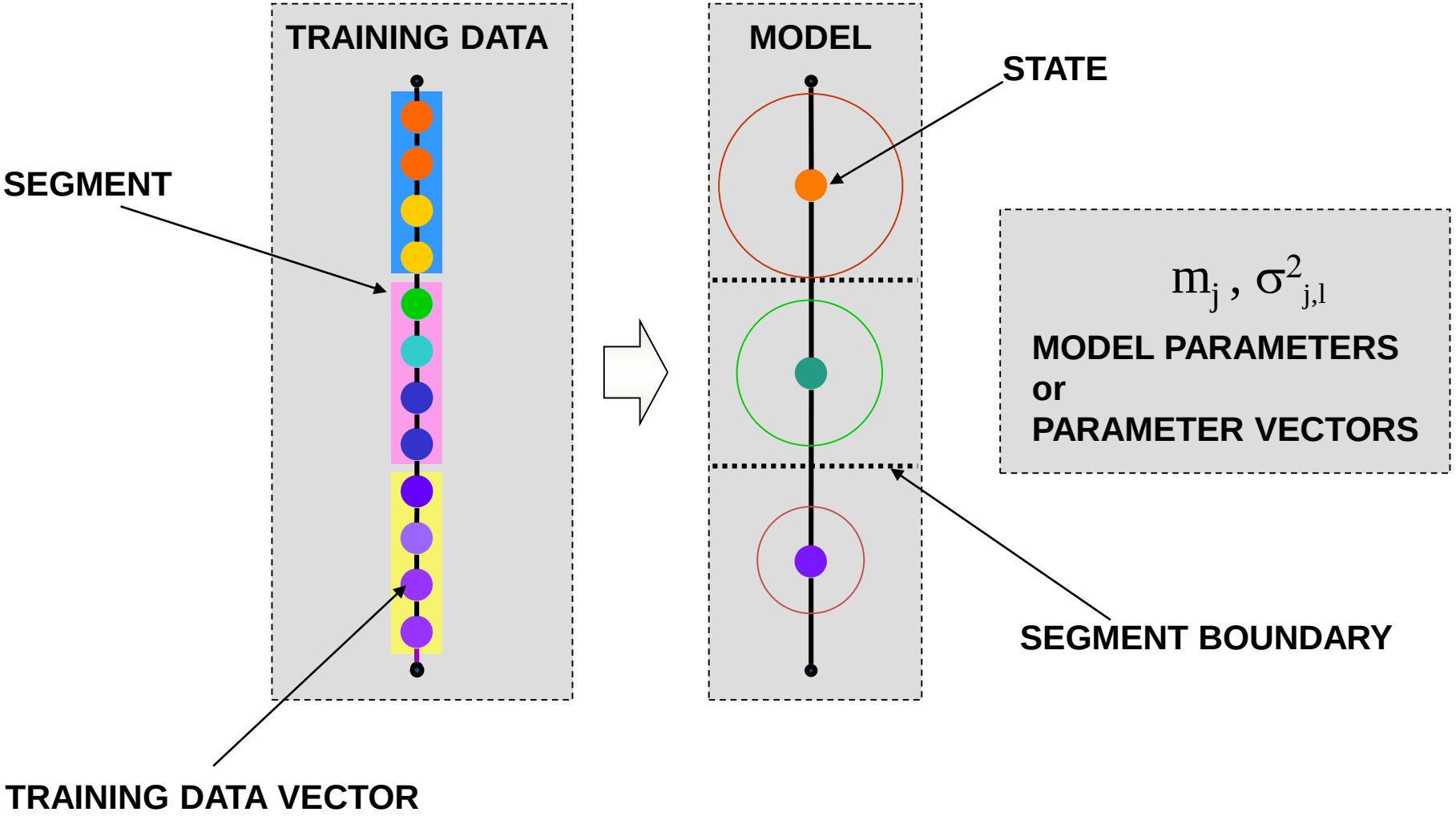
Alignment for training a model from multiple vector sequences



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

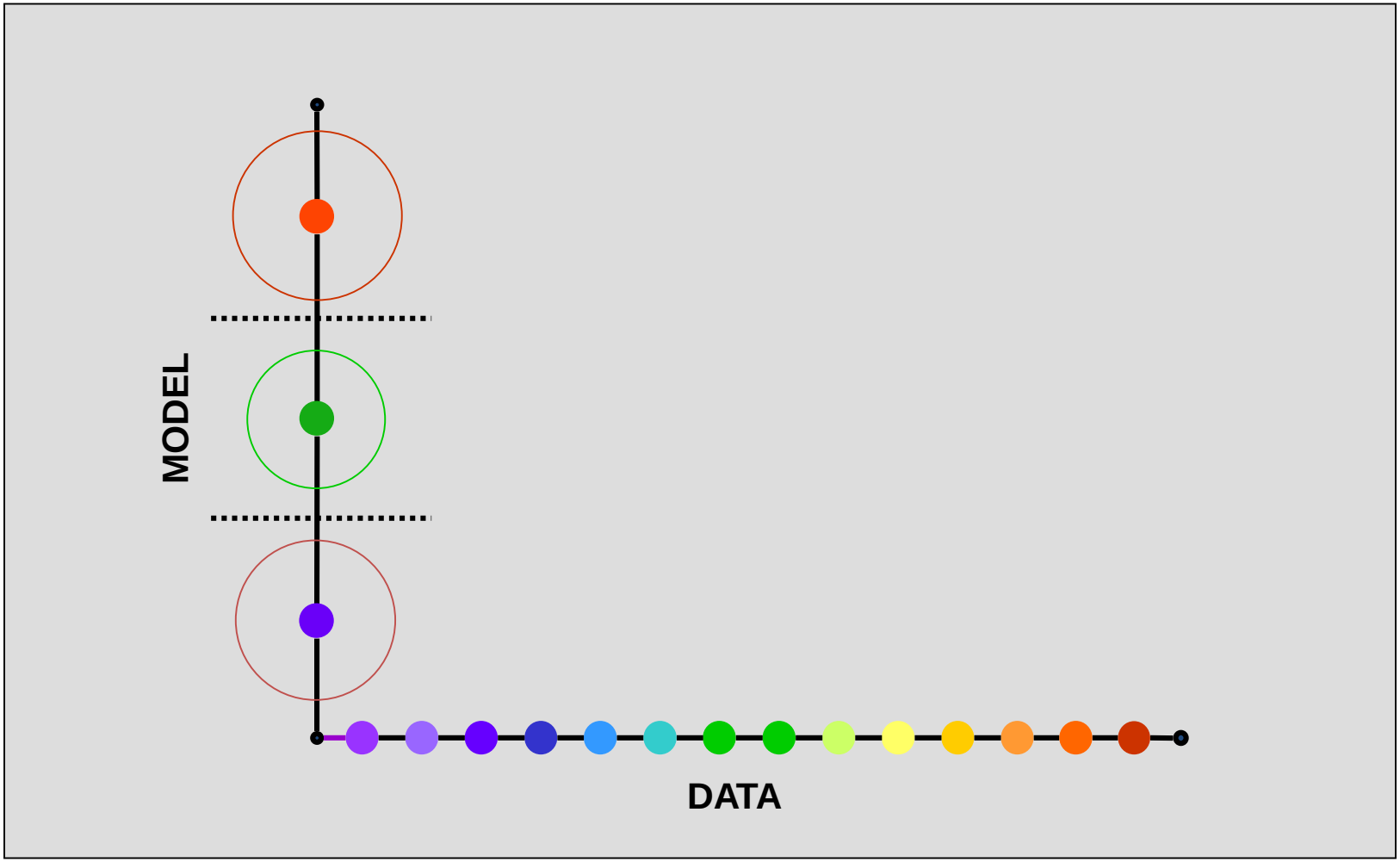
Shifted terminology



Improving the Templates

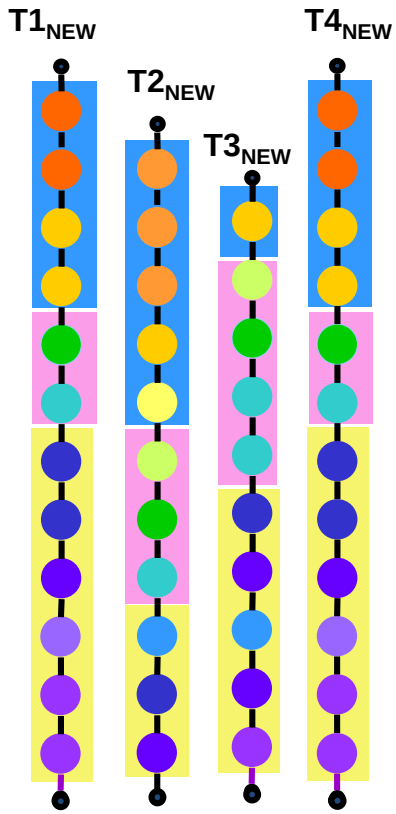
- Generalization by averaging the templates
- Generalization by reducing template length
- Accounting for variation within templates represented by the reduced model
- Accounting for varying segment lengths

Transition structures in models



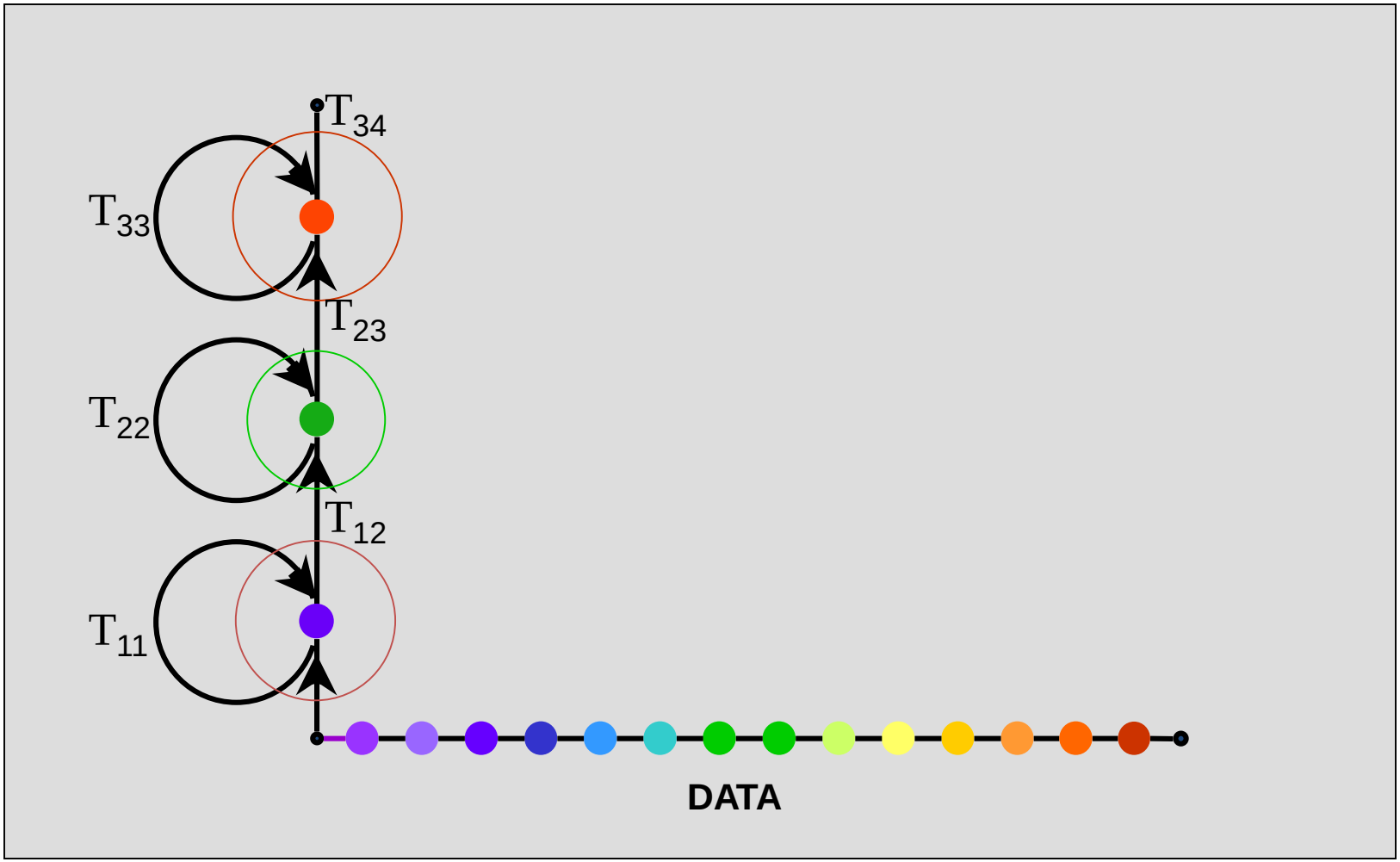
The converged models can be used to score / align data sequences
Model structure is incomplete.

DTW with multiple models



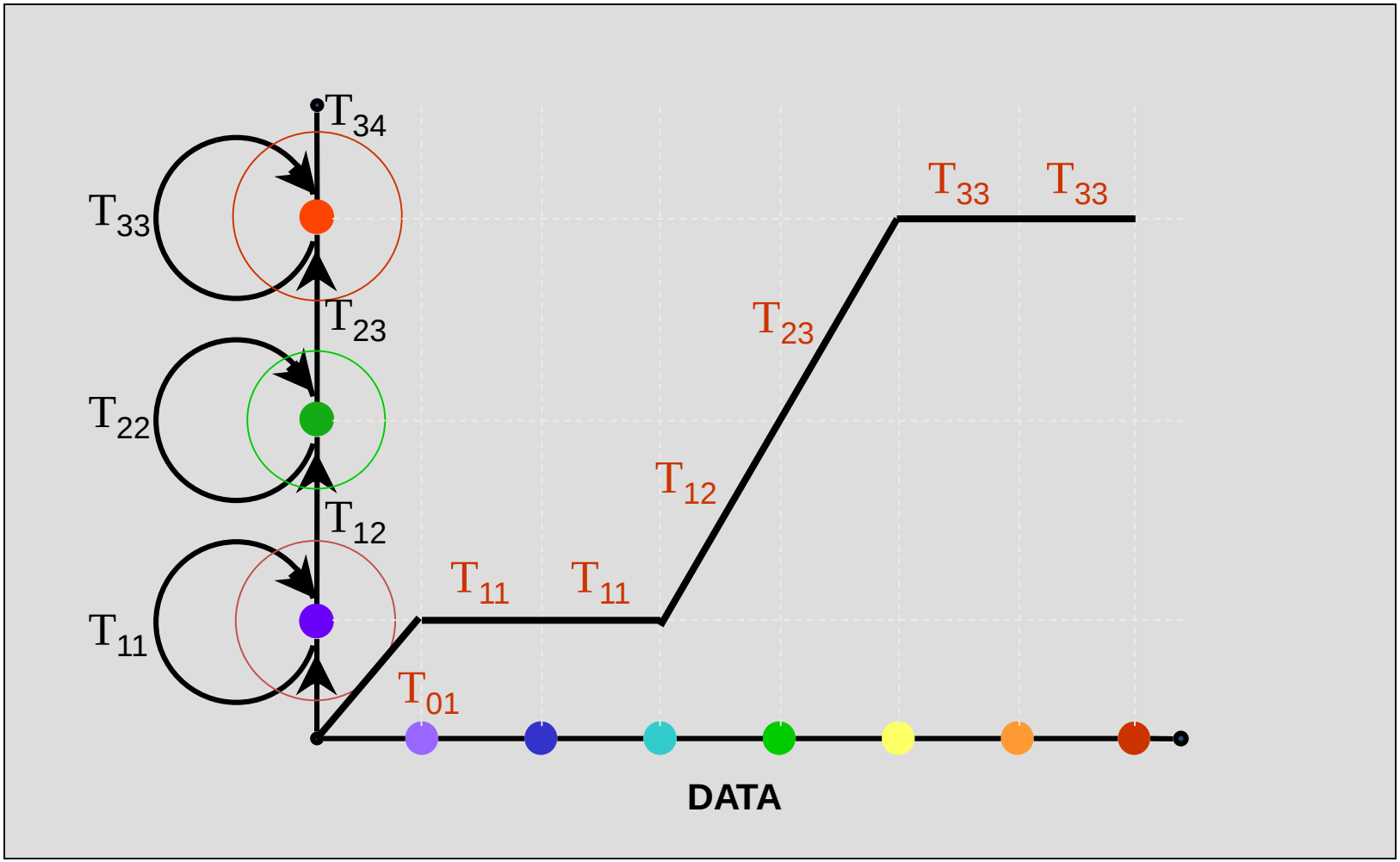
- Some segments are naturally longer than others
  - E.g., in the example the initial (yellow) segments are usually longer than the second (pink) segments
- This difference in segment lengths is different from the *variation* within a segment
  - Segments with small variance could still persist very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



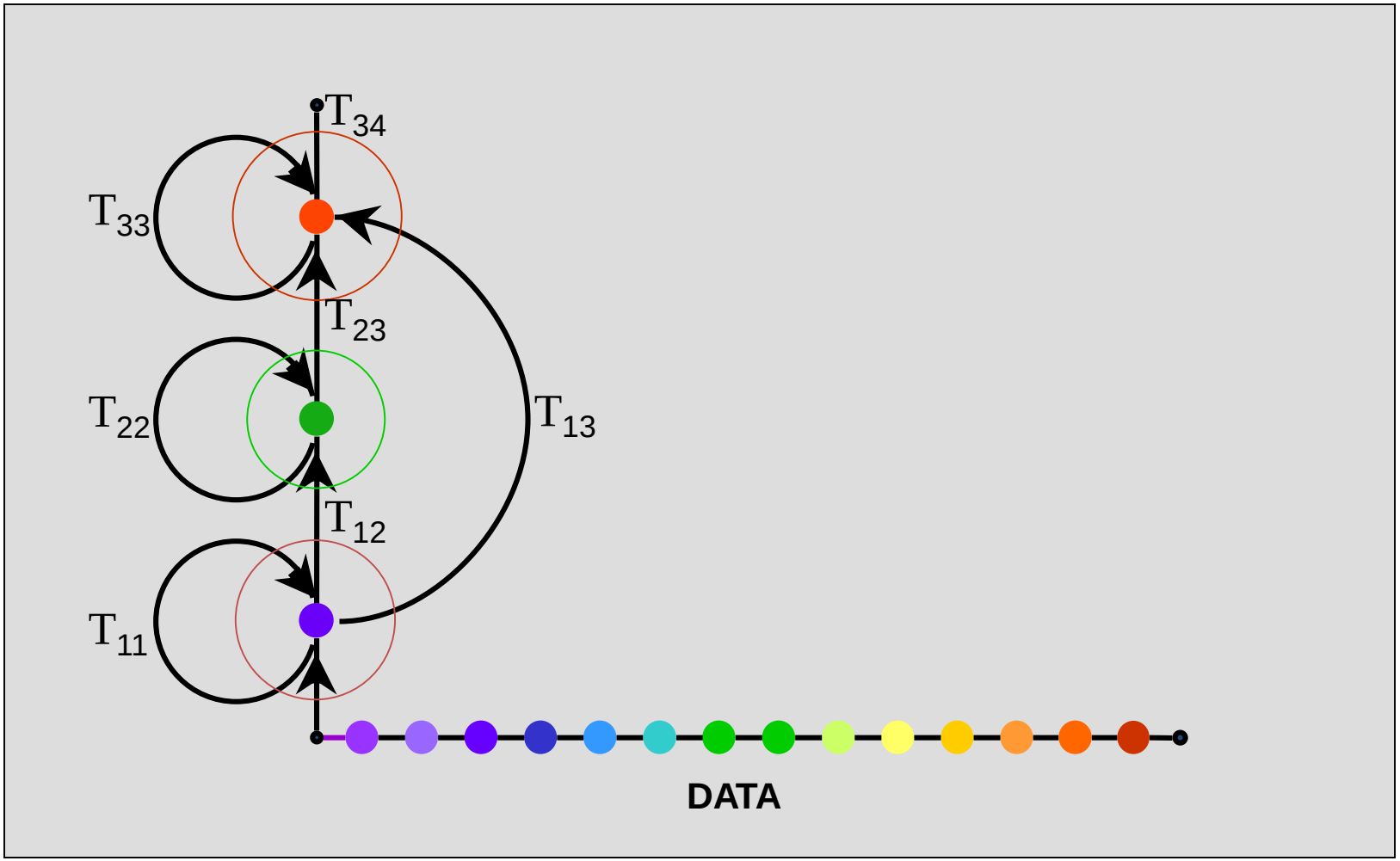
State specific insertion penalties are represented as self transition arcs for model vectors. Horizontal edges within the trellis will incur a penalty associated with the corresponding arc. Every transition within the model can have its own penalty.

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

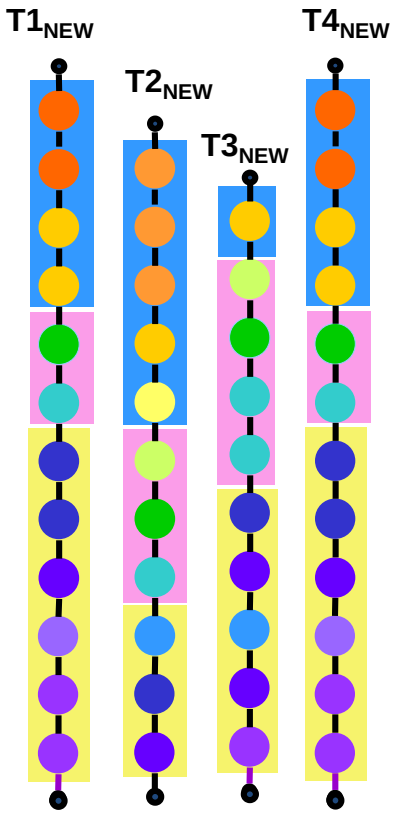
What should the transition scores be

- 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
  - If a vector in the i^{th} segment is typically followed by a vector in the j^{th} segment, but also rarely by vectors from 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

What should the transition scores be

- A good choice for transition scores is the negative logarithm of the probabilities of the 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

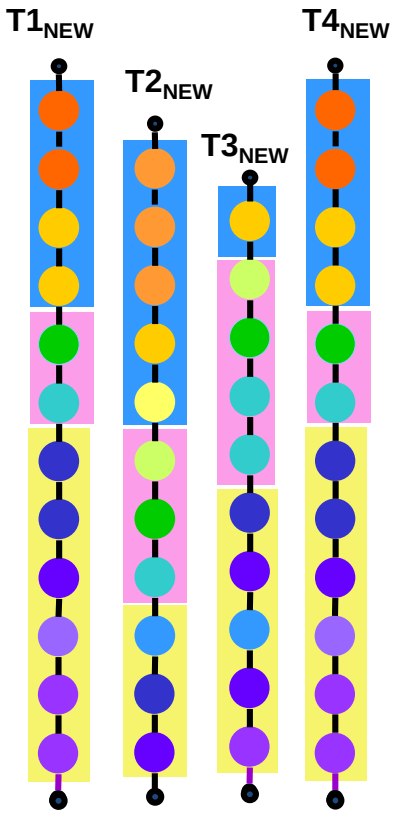


- Transition scores can be easily computed by a simple extension of the segmental K-means algorithm
- Probabilities can be counted by simple counting

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

- $N_{k,i}$ 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



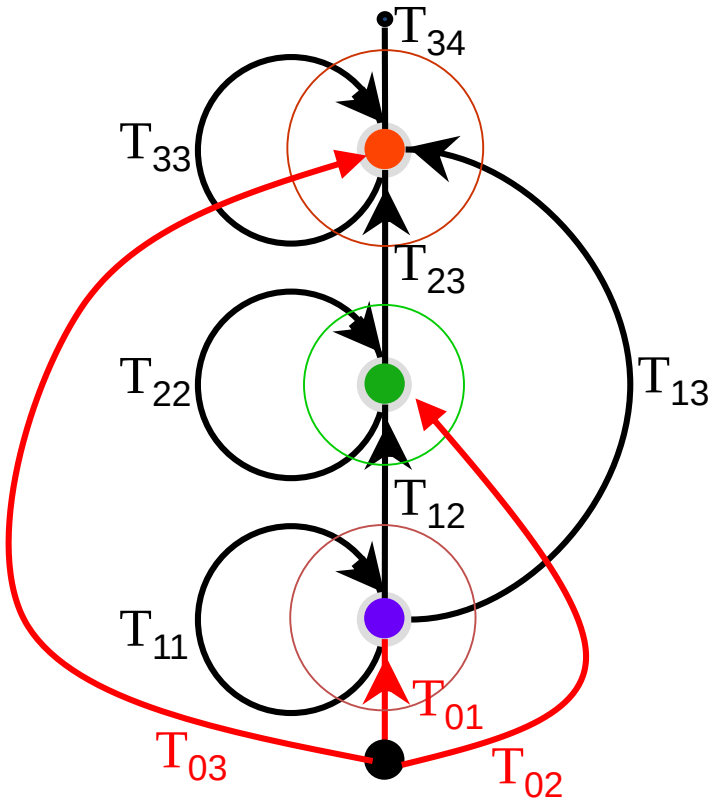
N = 4
N_{01} = 4
N_{02} = 0
N_{03} = 0

- 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

What is the Initial State Probability?



Transition from an initial “dummy” state

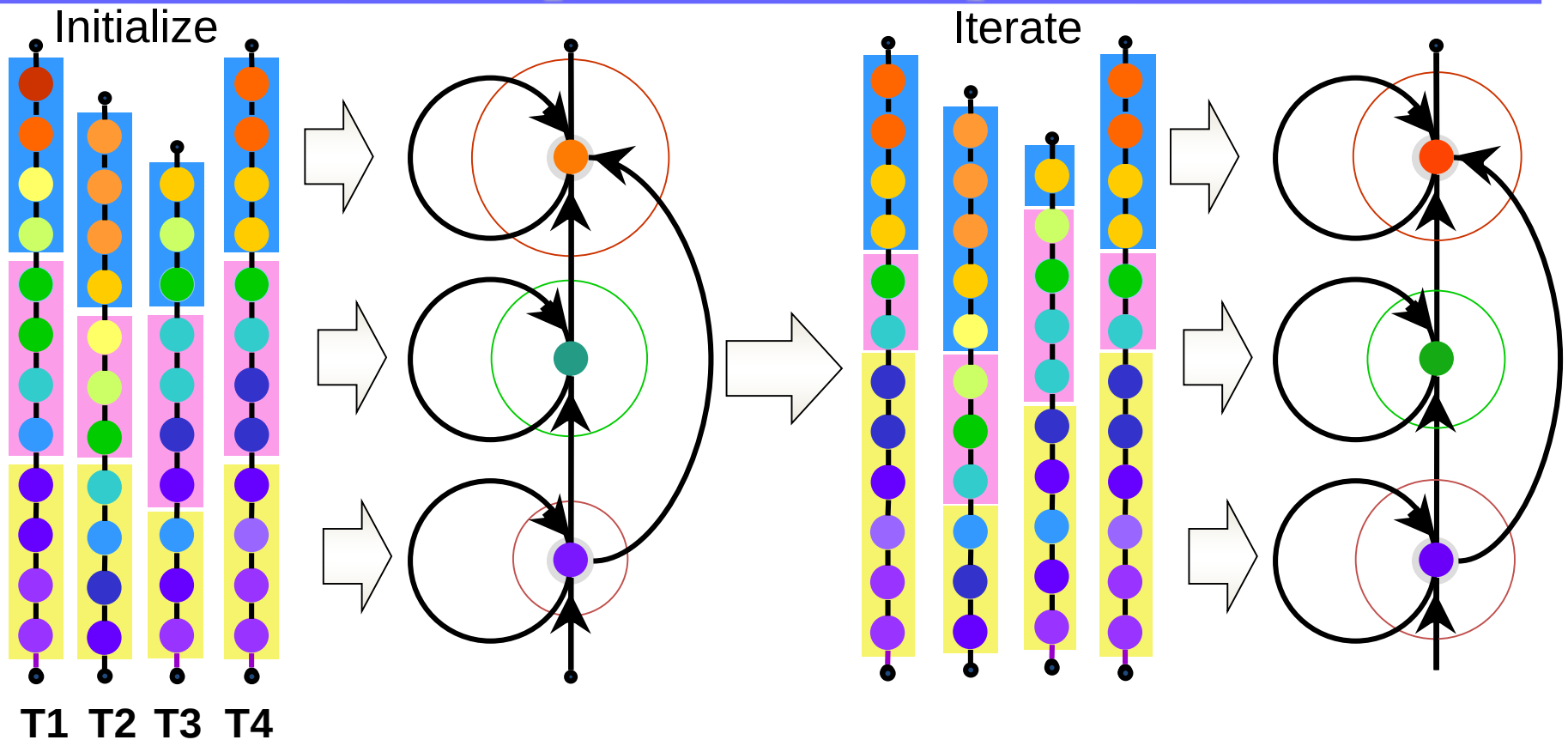
Modified segmental K-means AKA Viterbi training

- 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

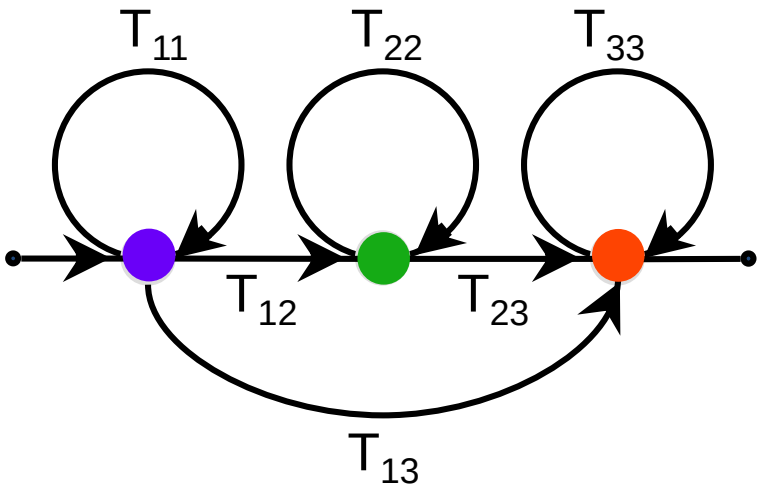


The procedure can be continued until convergence

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

The resulting model structure is also known as an HMM!

DTW and Hidden Markov Models (HMMs)



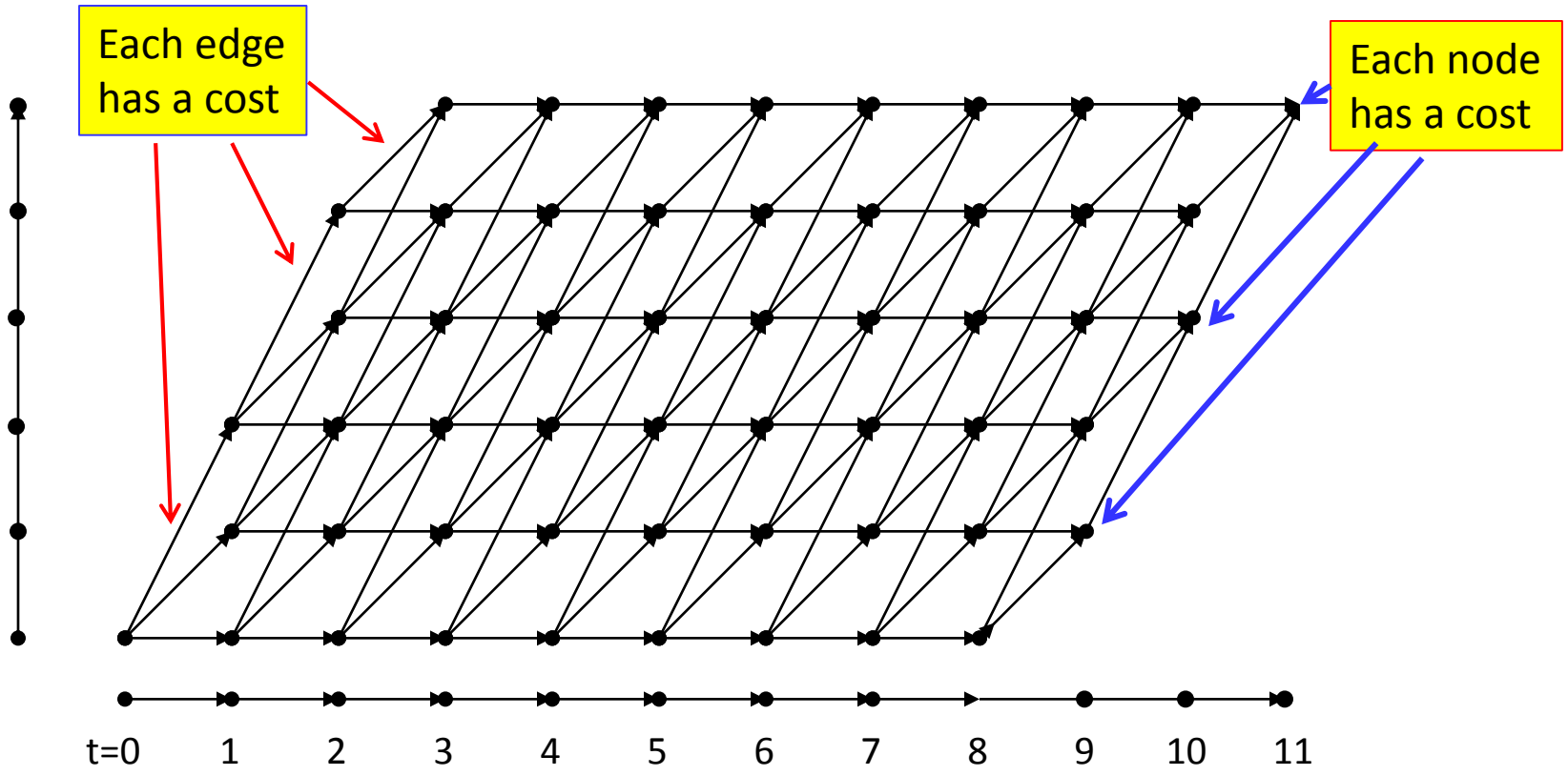
- This structure is a generic representation of a statistical model for processes that generate time series
- The "segments" in the time series are referred to as states
  - The process passes through these states to generate time series
- The entire structure may be viewed as *one* generalization of the DTW models we have discussed thus far
- In this example -- strict left-to-right topology
  - Commonly used for speech recognition

DTW -- Reversing Sense of "Cost"

- Use "Score" instead of "Cost"
  - The same cost function but with the sign changed
  - *E.g.* for at a node: *negative* Euclidean distance $= -\sqrt{\Sigma(x_i - y_i)^2}$;
  - $-\Sigma(x_i - y_i)^2$; *i.e.* –ve Euclidean distance squared
  - Other terms possible:
    - Remember the Gaussian

DTW with *costs*

- Node and edge *costs* defined for trellis
- Find minimum cost path through trellis..

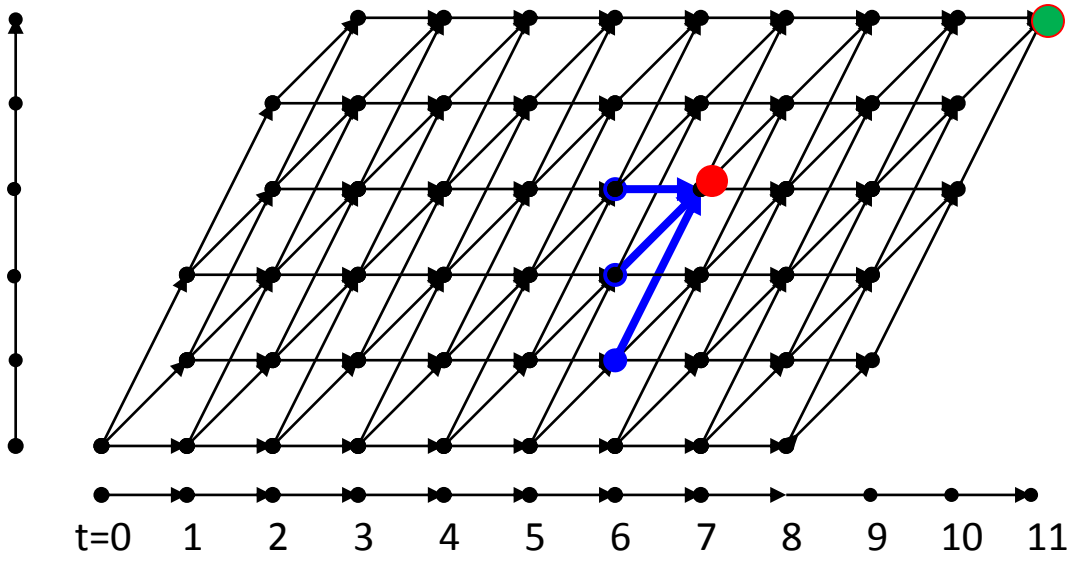


DTW: Finding minimum-cost path

- $P_{i,j}$ = best path cost from origin to node [i,j]
- $C_{i,j}$ = local node cost of aligning template frame i to input frame j
- $T_{i,j,k,l}$ = Edge cost from node (i,j) to node (k,l)

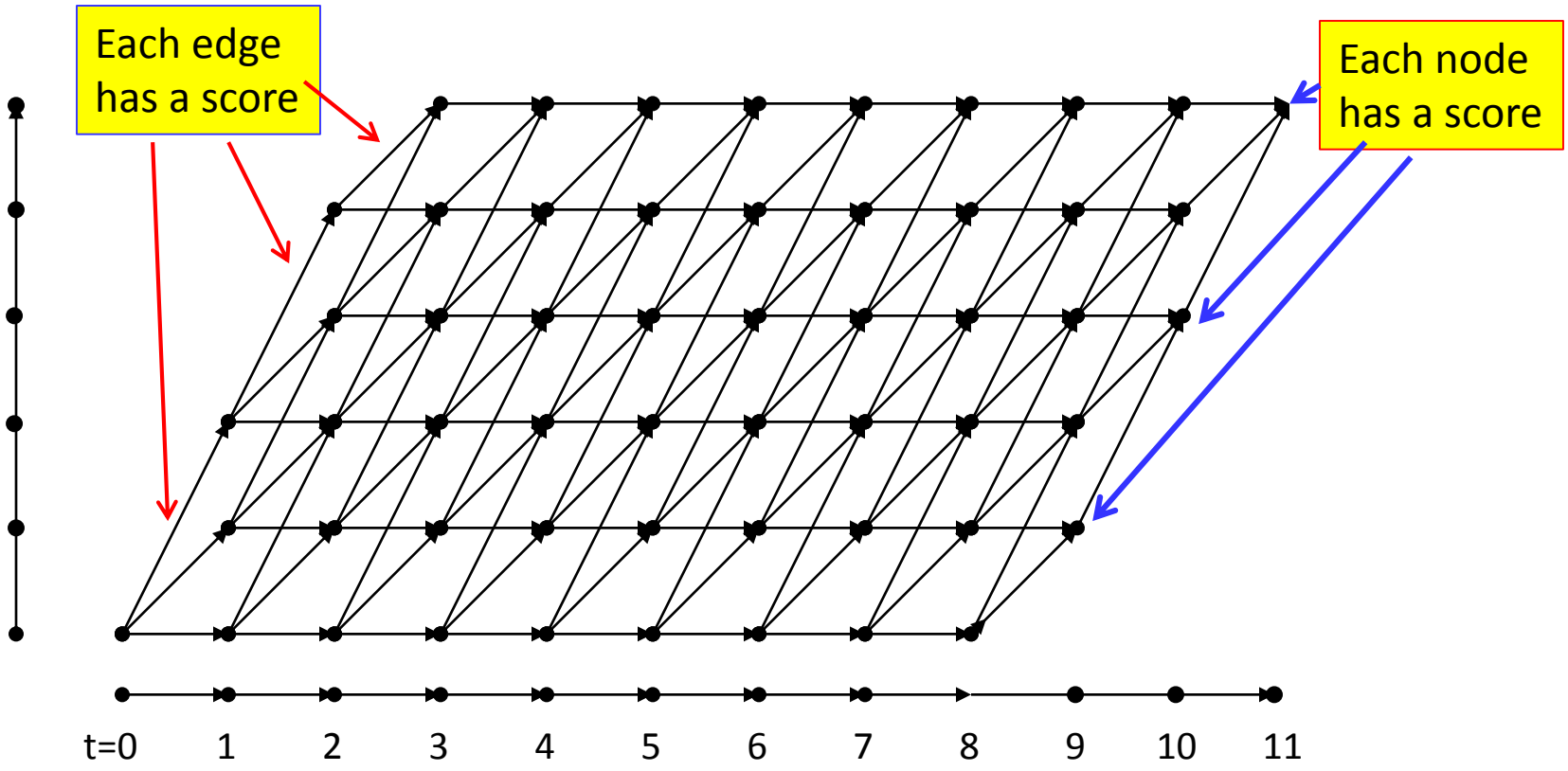
$$P_{i,j} = \min (P_{i,j-1} + T_{i,j-1,i,j} , P_{i-1,j-1} + T_{i-1,,j-1,i,j} , P_{i-2,j-1} + T_{i-2,j-1,i,j} ,) + C_{i,j}$$

  - **MINIMIZE TOTAL PATH COST**



DTW with **scores**

- Node and edge *scores* defined for trellis
- Find maximum score path through trellis..

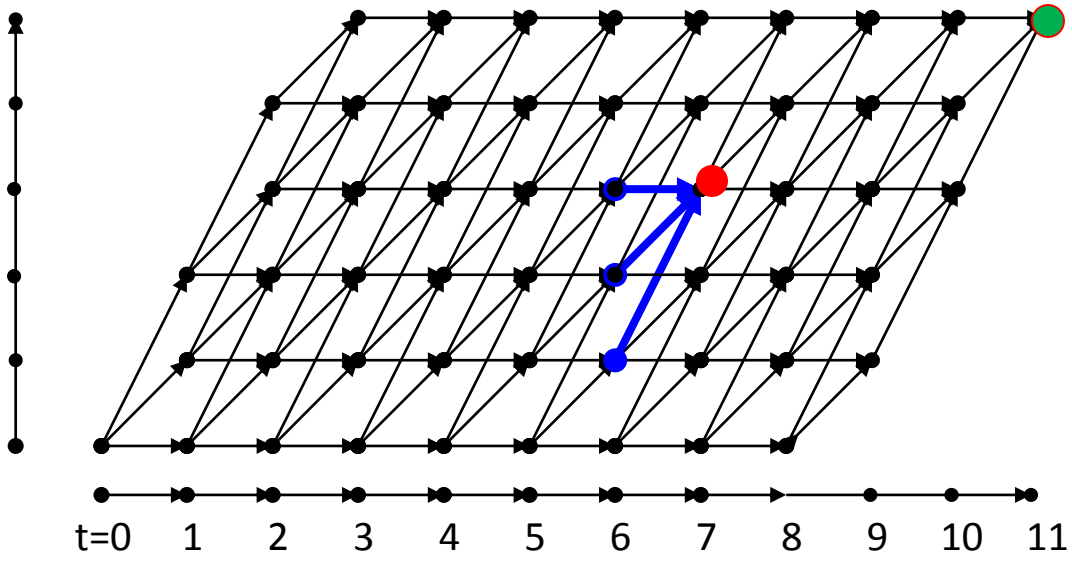


DTW: finding ***maximum-score*** path

- $P_{i,j}$ = best path score from origin to node [*i*,*j*]
- $C_{i,j}$ = local node score of aligning template frame *i* to input frame *j*
- $T_{i,j,k,l}$ = Edge score from node (i,j) to node (k,l)

$$P_{i,j} = \max\left(P_{i,j-1} + T_{i,j-1,i,j},\ P_{i-1,j-1} + T_{i-1,,j-1,i,j},\ P_{i-2,j-1} + T_{i-2,j-1,i,j},\right) + C_{i,j}$$

- **MAXIMIZE TOTAL PATH SCORE**



Probabilities for Scores

- HMM – inference equivalent to DTW modified to use a *probabilistic* function, for the local node or edge "scores" in the trellis
  - Edges have transition *probabilities*
  - Nodes have *output* or *observation probabilities*
    - They provide the probability of the observed input
    - The output probability may be a Gaussian
  - The goal is to find the template with highest probability of matching the input

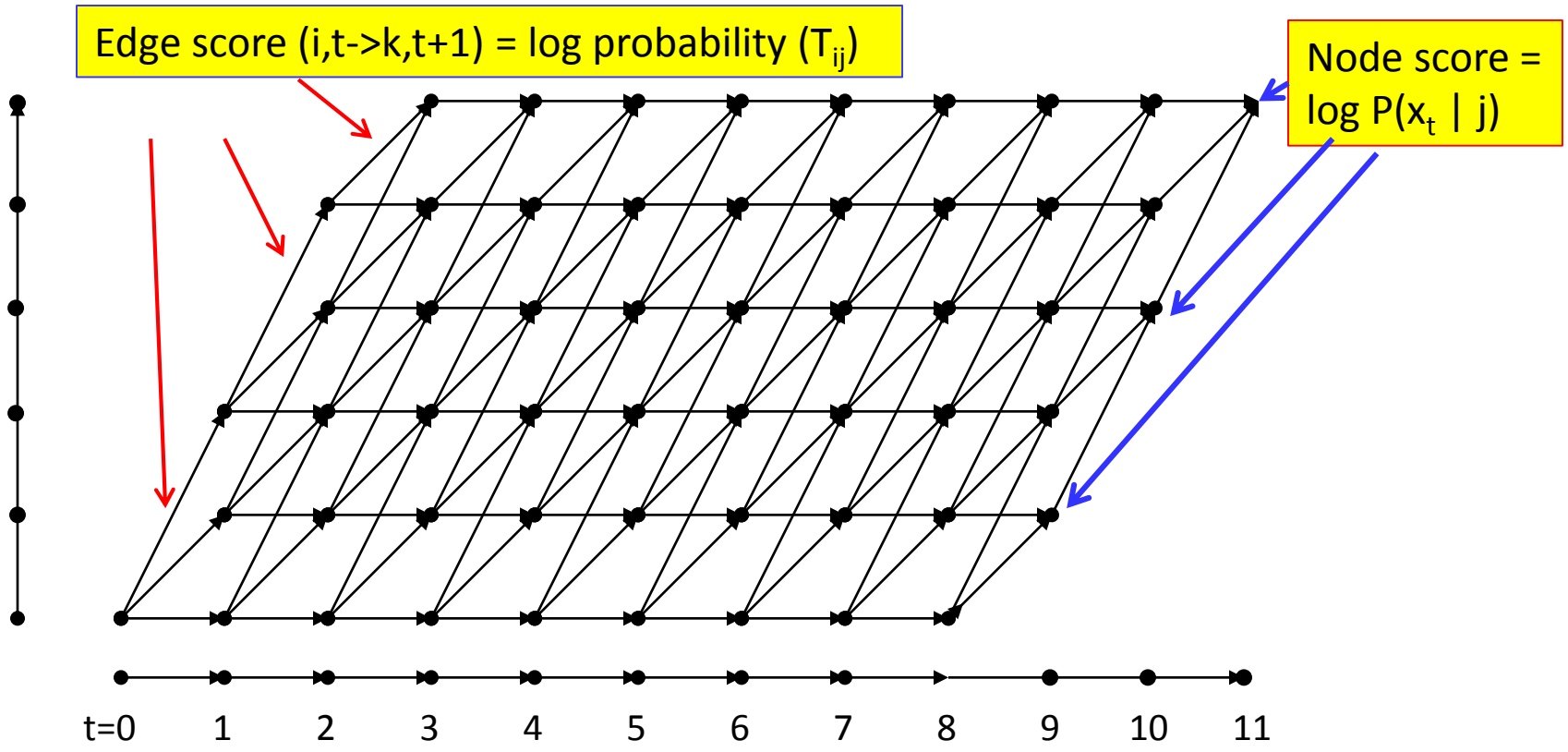
- Probability values as "scores" are also called *likelihoods*

Log Likelihoods

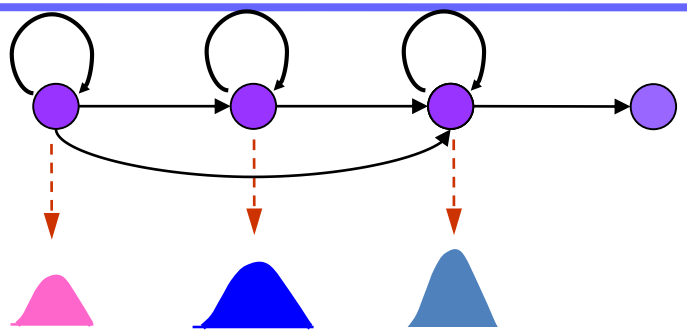
- The problem with probabilities: The probabilities of unrelated variables multiply
  - P(X,Y) = P(X)P(Y)
- Probabilities multiply along the path
  - Scores combines multiplicatively along a path
  - score of a path = Product_over_nodes(score of node) x Product_over_edges(score of edge)
- Use *log* probabilities as scores
  - Scores add as in DTW
    - Max instead of Min
- May use *negative* log probabilities
  - *Cost* adds as in DTW

HMM = DTW with **scores**

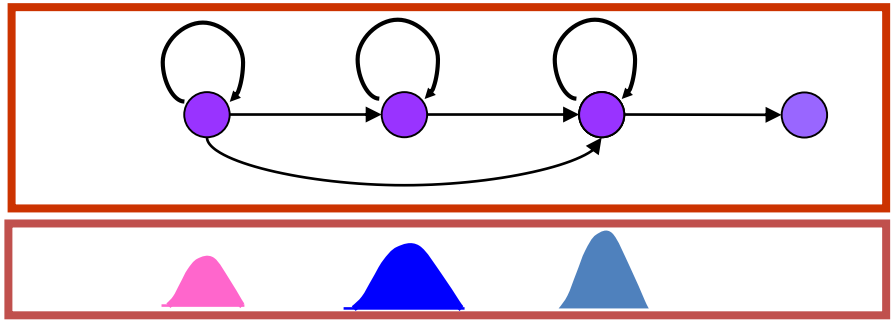
- Node and edge *scores* are log probabilities
- Find maximum score path through trellis..



Hidden Markov Models



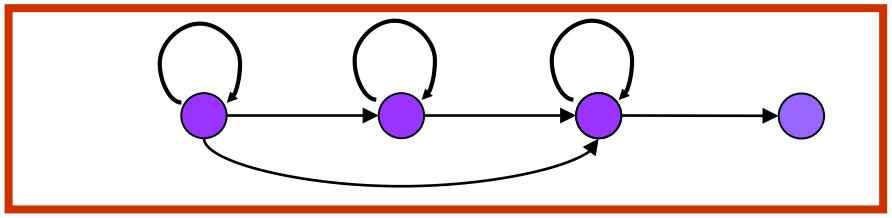
- A Hidden Markov Model consists of two components
  - A state/transition backbone that specifies how many states there are, and how they can follow one another
  - A set of probability distributions, one for each state



- 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

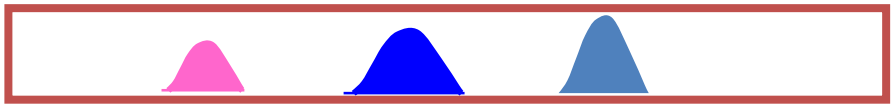
Basic Structural Questions for HMM

- What is the structure like?
  - What transitions are allowed
  - How many states?



Markov chain

- What are the probability distributions associated with states?



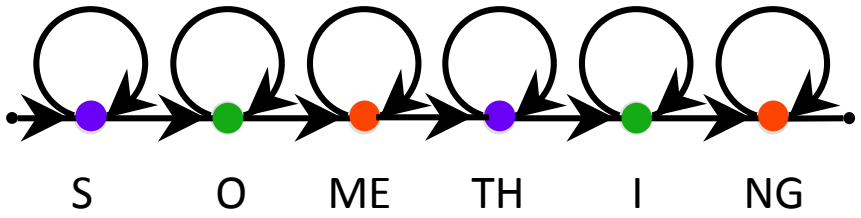
Data distributions

Determining the Number of States

- The correct number of states for any word?
  - We do not know, really
  - Ideally there should be at least one state for each "basic sound" within the word
    - Otherwise widely differing sounds may be collapsed into one state
  - For efficiency, the number of states should the minimum needed to achieve the desired level of recognition accuracy
  - These two are conflicting requirements, usually solved by making some educated guesses

Determining the Number of States

- For small vocabularies, it is possible to examine each word in detail and arrive at reasonable numbers:

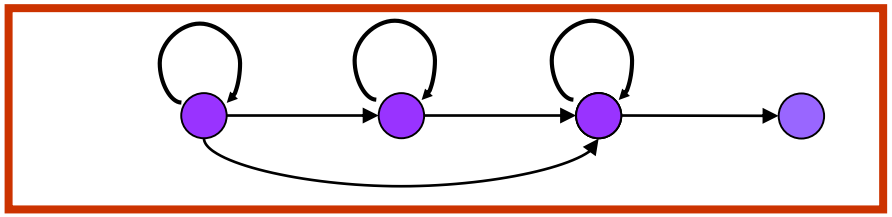


- For larger vocabularies, we may be forced to rely on some *ad hoc* principles
  - *E.g.* proportional to the number of letters in the word
    - Works better for some languages than others
    - Spanish, Japanese (Katakana/Hiragana), Indian languages..

What about the transition structure

- Speech is a left to right process
  - With a deterministic structure to any word
    - Prespecified set of sounds in prespecified order
  - Although some sounds may occasionally be skipped

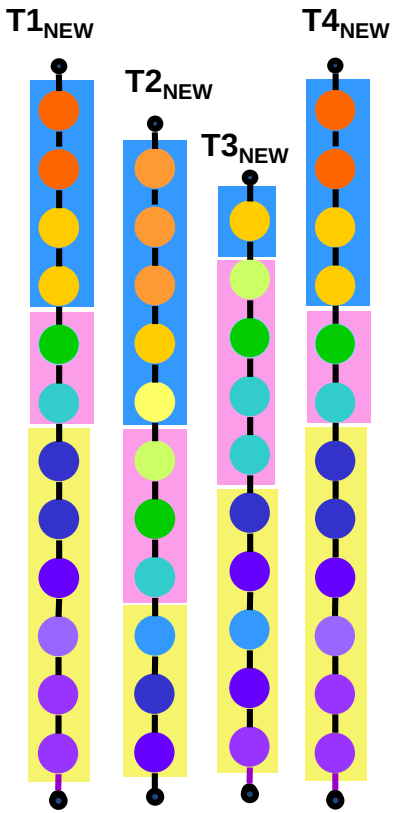
- This suggests the following kind of structure



Markov chain

- The *Bakis* topology

The Transition *Probabilities?*



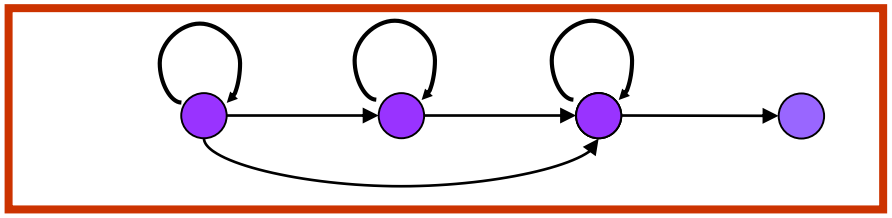
- Transition probabilities are just the probability of making transitions
- As before, can be obtained by simple counting

$$P_{ij} = \frac{\sum_k N_{k,i,j}}{\sum_k N_{k,i}}$$

- $N_{k,i}$ 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)

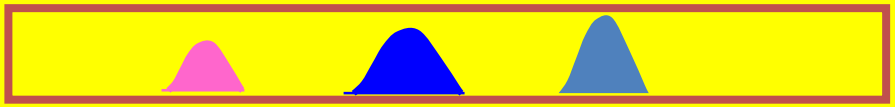
Basic Structural Questions for HMM

- What is the structure like?
  - What transitions are allowed
  - How many states?



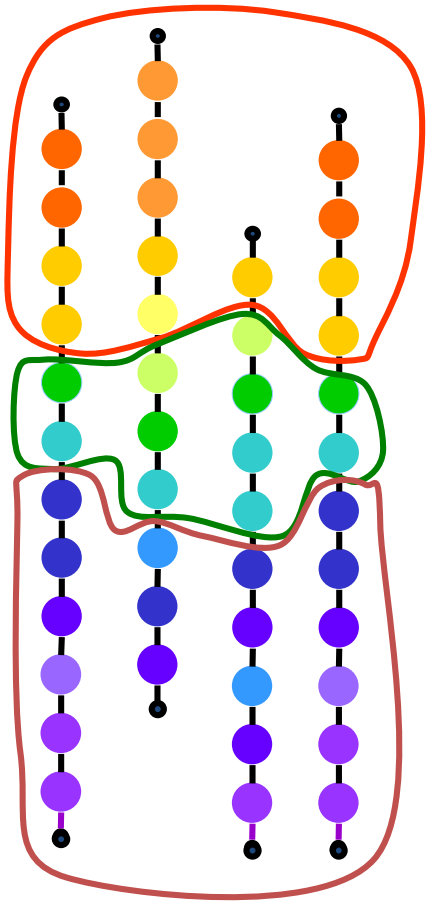
Markov chain

- What are the probability distributions associated with states?



Data distributions

State Output Distributions



- The state output distribution for any state is the distribution of all vectors associated with that state
- It is the distribution of *all* vectors validly associated with the state
  - From all potential instances of the word
  - Not just the training (template) recordings
- We have implicitly assumed this to be Gaussian so far

The State Output Distribution

- The state output distribution is a probability distribution associated with each HMM state
  - The negative log of the probability of any vector as given by this distribution would be the node cost in DTW
- The state output probability distribution could be any distribution at all
- We have assumed so far that state output distributions are Gaussian

$$P(x \mid j) = \frac{1}{\sqrt{\prod_l 2\pi\sigma_{j,l}^2}} \exp\left(-0.5 \sum_l \frac{(x_l - m_{j,l})^2}{\sigma_{j,l}^2} \right)$$

$$d_j(v) = -\log(P(x \mid j)) = 0.5 \sum_l \log(2\pi\sigma_{j,l}^2) + 0.5 \sum_l \frac{(x_l - m_{j,l})^2}{\sigma_{j,l}^2}$$

Node cost for DTW (note change in notation)

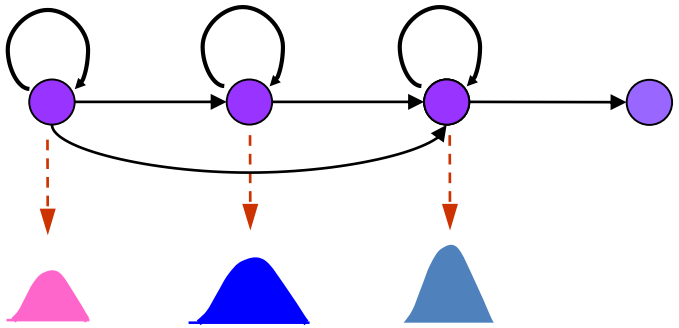
- **More generically, we can assume it to be a *mixture* of Gaussians**

$$P(x \mid j) = \sum_k \frac{w_k}{\sqrt{\prod_l 2\pi\sigma_{j,k,l}^2}} \exp\left(-0.5 \sum_l \frac{(x_l - m_{j,k,l})^2}{\sigma_{j,k,l}^2} \right)$$

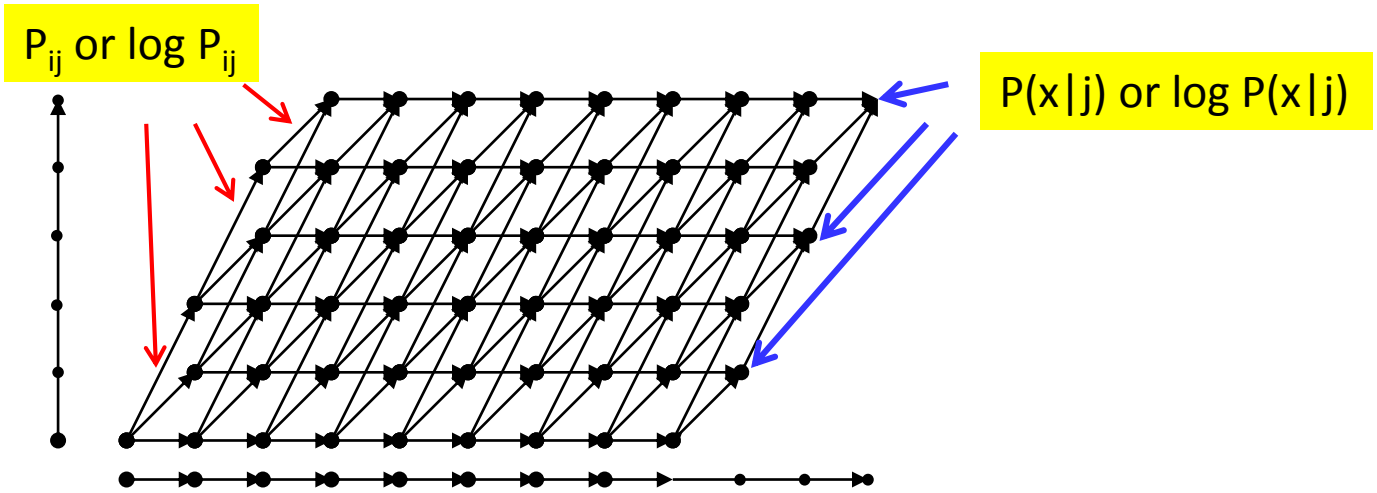
$$d_j(v) = -\log(P(x \mid j))$$

- More on this later

The complete package



- We have seen how to compute all probability terms for this structure
- Sufficient information to compute all terms in a trellis



We continue in the next class..