Hidden Markov Models for Speech Recognition

Bhiksha Raj

22 Feb 2010

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

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
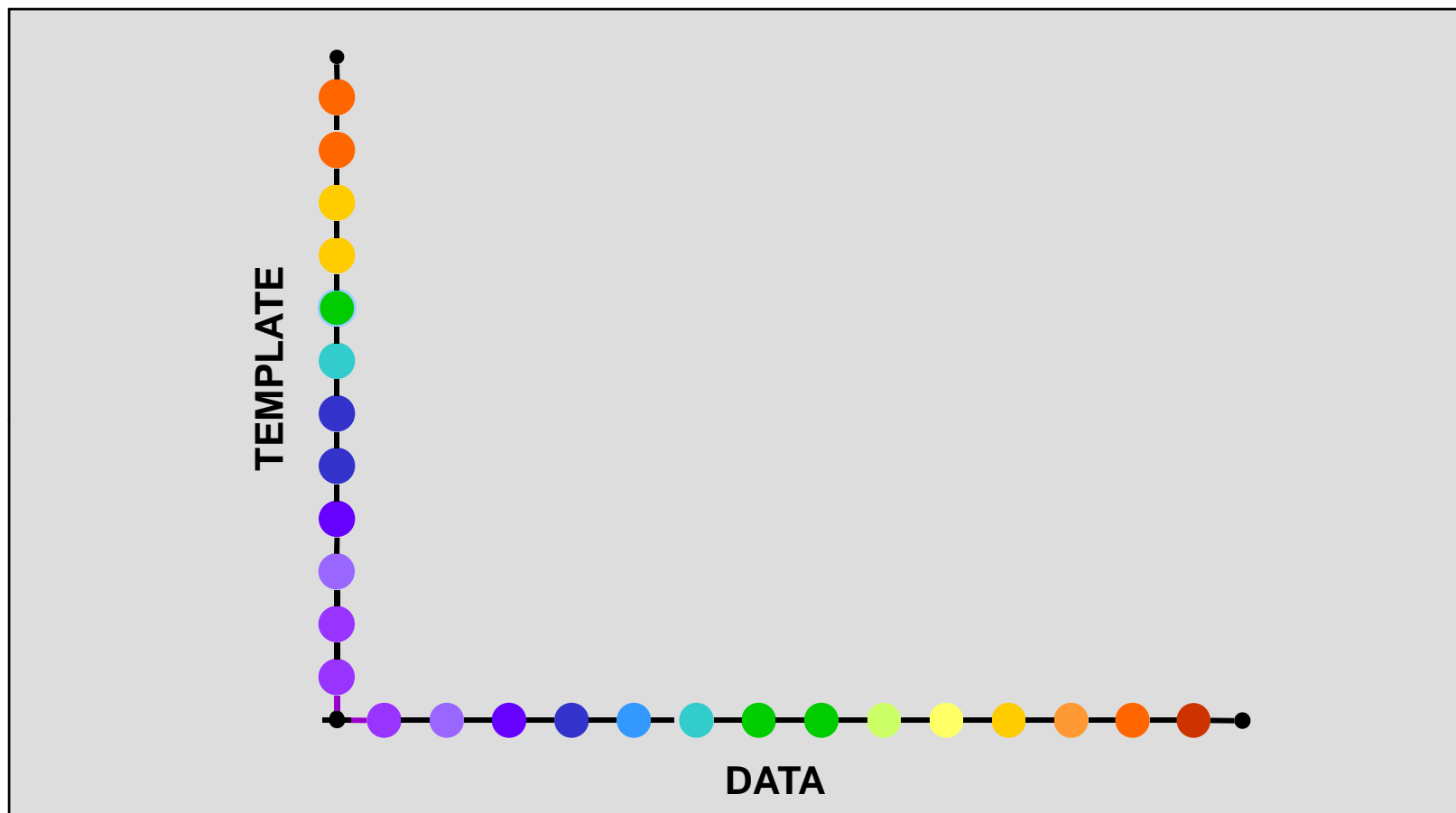
We've seen the DTW alignment of data to model

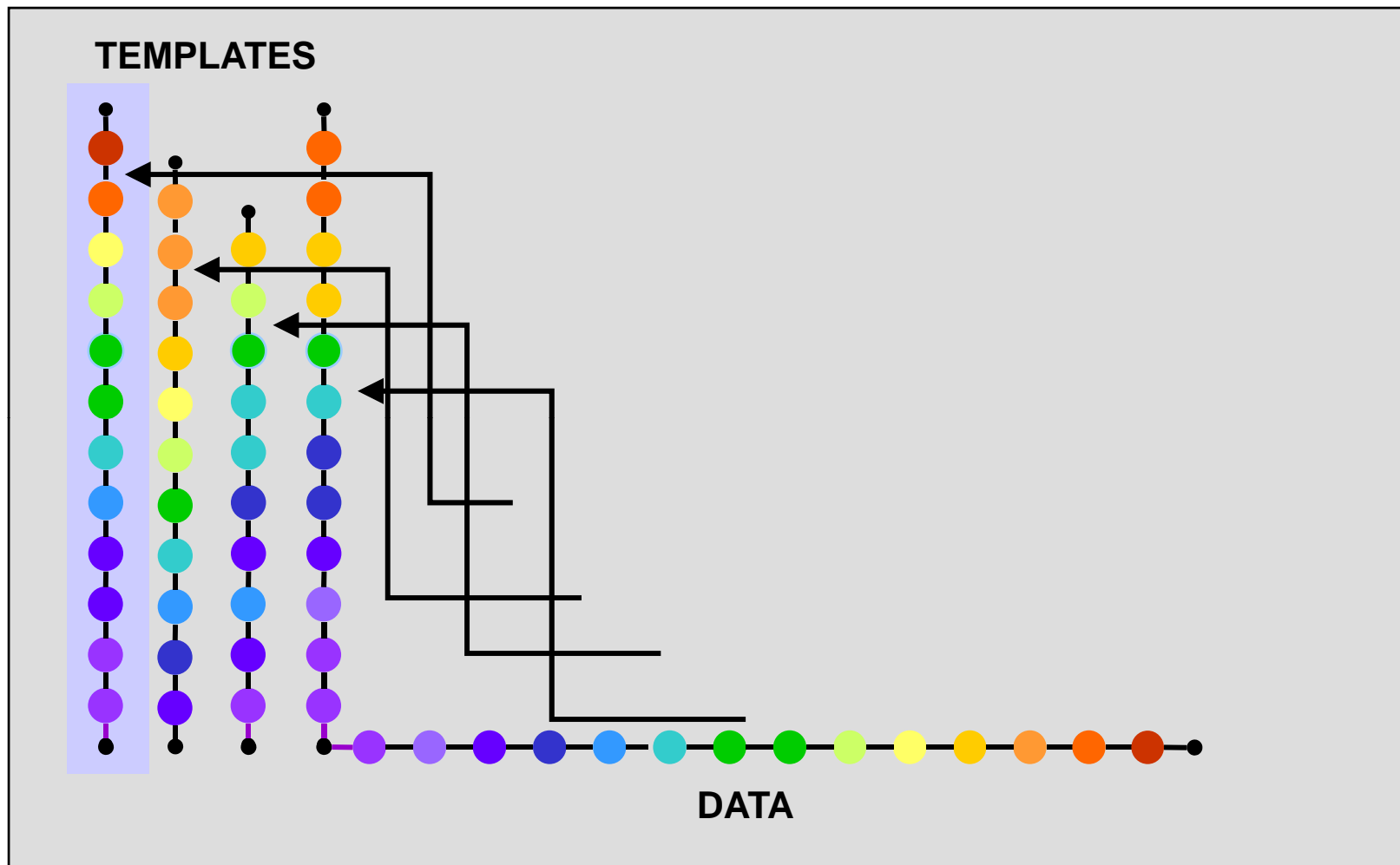(*model axis inverted from earlier discussion*)

Limitations of A Single Template

☐ As noted in the previous topic, a single template cannot capture all the variations in speech

☐ One alternative already suggested: use multiple templates for each word, and match the input against each one

DTW with multiple templates
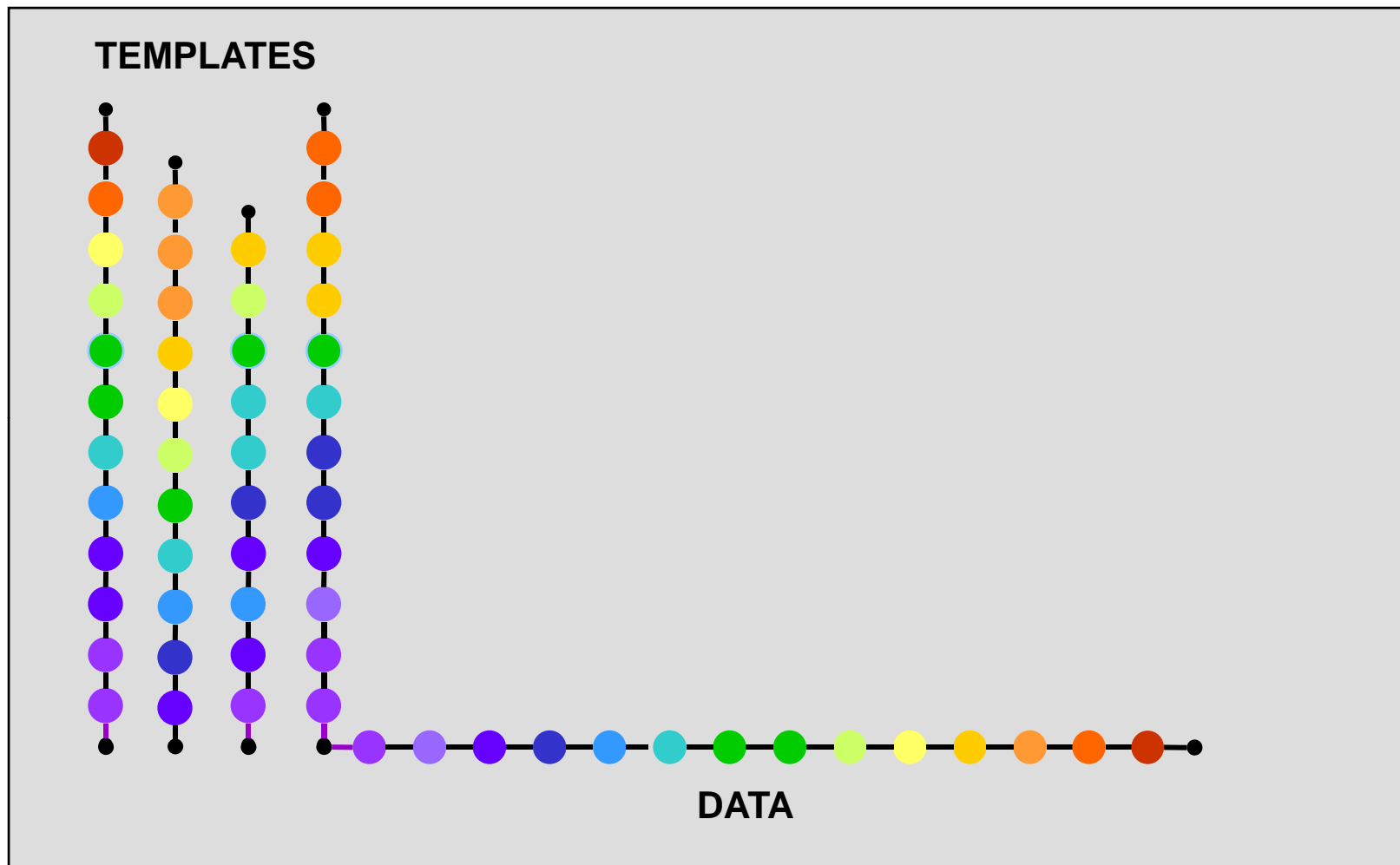
DTW with multiple templates

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

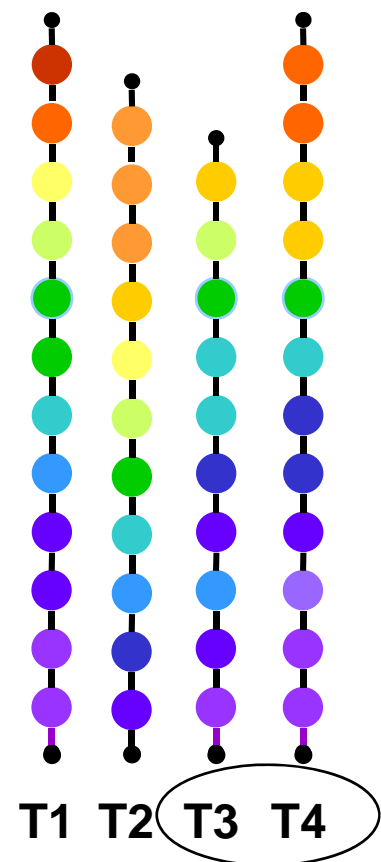- ☐ We will consider several steps to accomplish this

Improving the Templates

☐ Generalization by averaging the templates

☐ Generalization by reducing template length

☐ Accounting for variation within templates represented by the reduced model

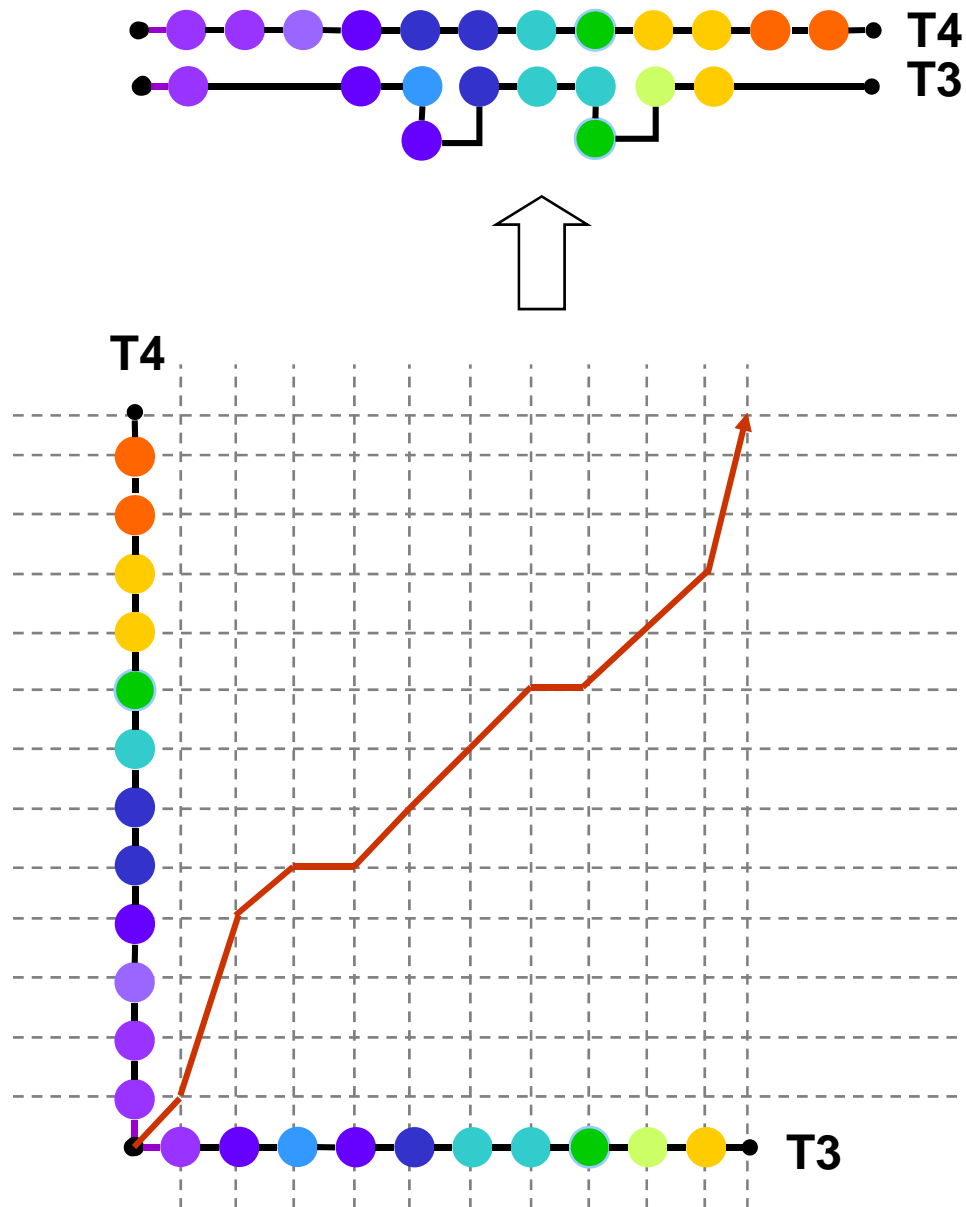☐ Accounting for varying segment lengths

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
    - ☐ *Note*: This requires not just finding the best cost, but the actual alignment between the template and input frame sequences, using the *back-pointers* described earlier
  - ■ Use the alignments so 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?
    - ☐ No definitive answer exists
    - ☐ Only trial and error solutions exist
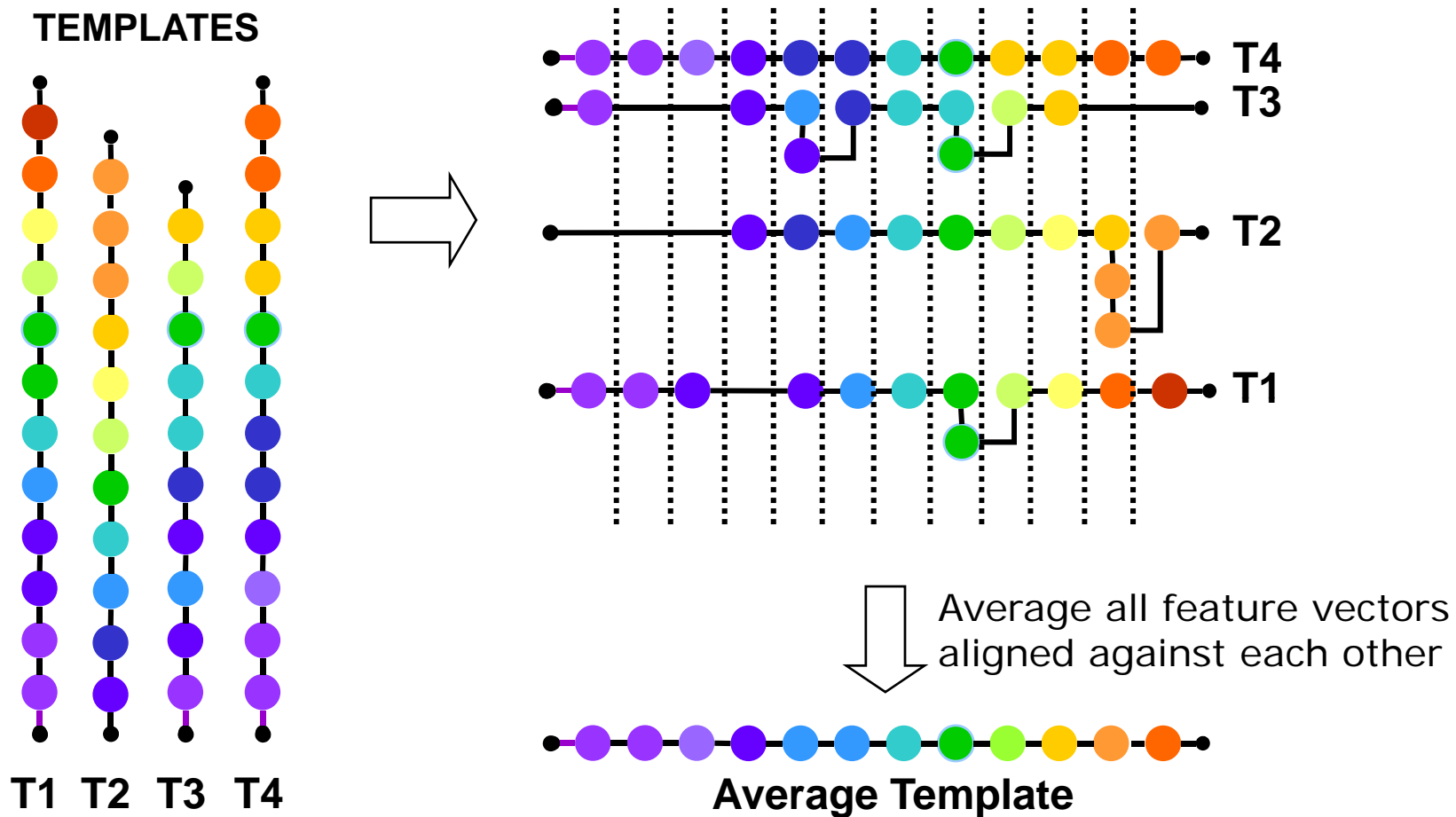
DTW with multiple templates

TEMPLATES

T1 T2 T3 T4

Align T4 and T3

T4

T3

12

DTW with multiple templates

TEMPLATES

T1 T2 T3 T4

T4

T3

T2

T1

Average all feature vectors aligned against each other

Average Template

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

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

Benefits of Template Averaging

- Obviously, 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, and more generally, assumed to be representative of future utterances of the word

- The more the number of templates, the better the generalization

15

Template Size Reduction

☐ Can we do better? Consider the template for "something":
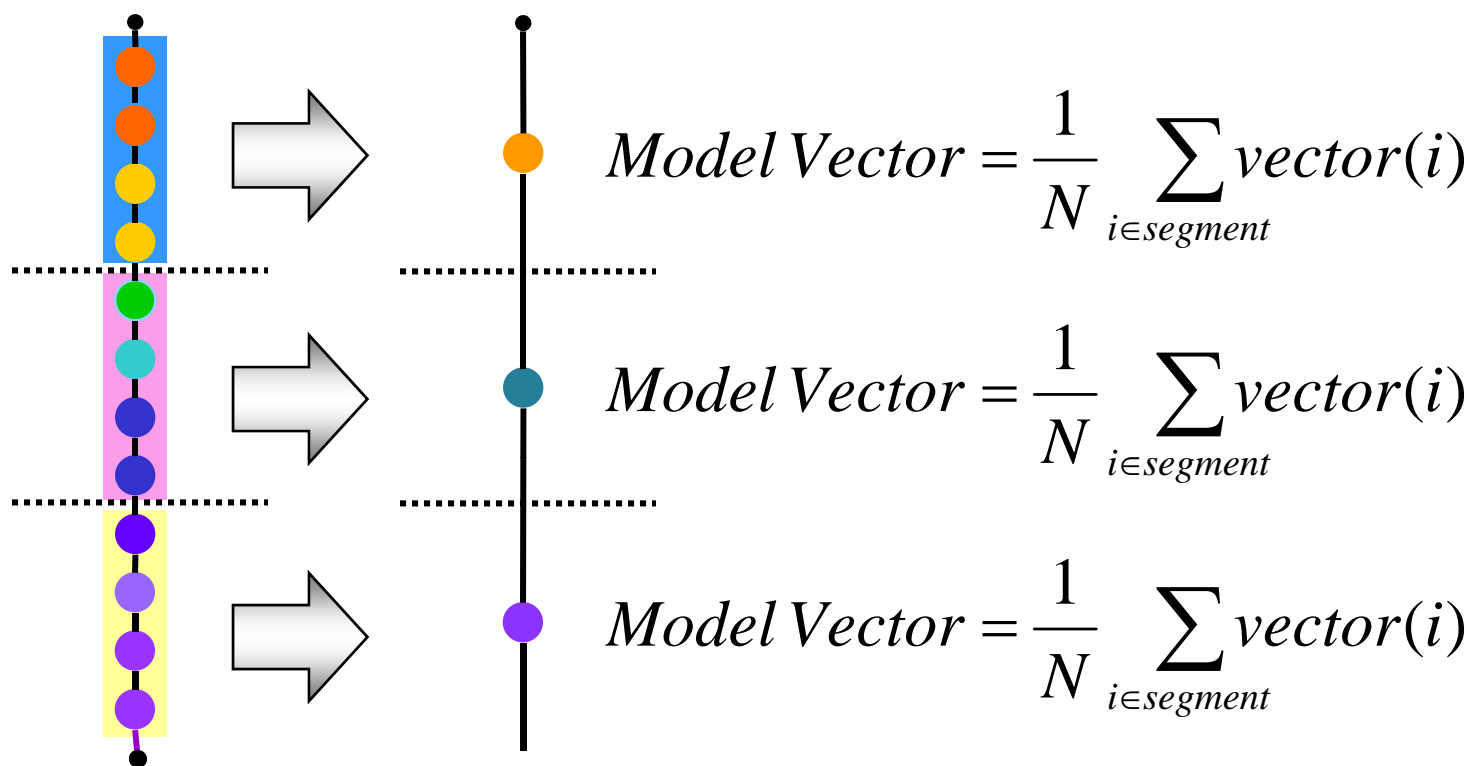
template | s | o | me | th | i | ng |

☐ 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

TEMPLATE

Three segments

DATA
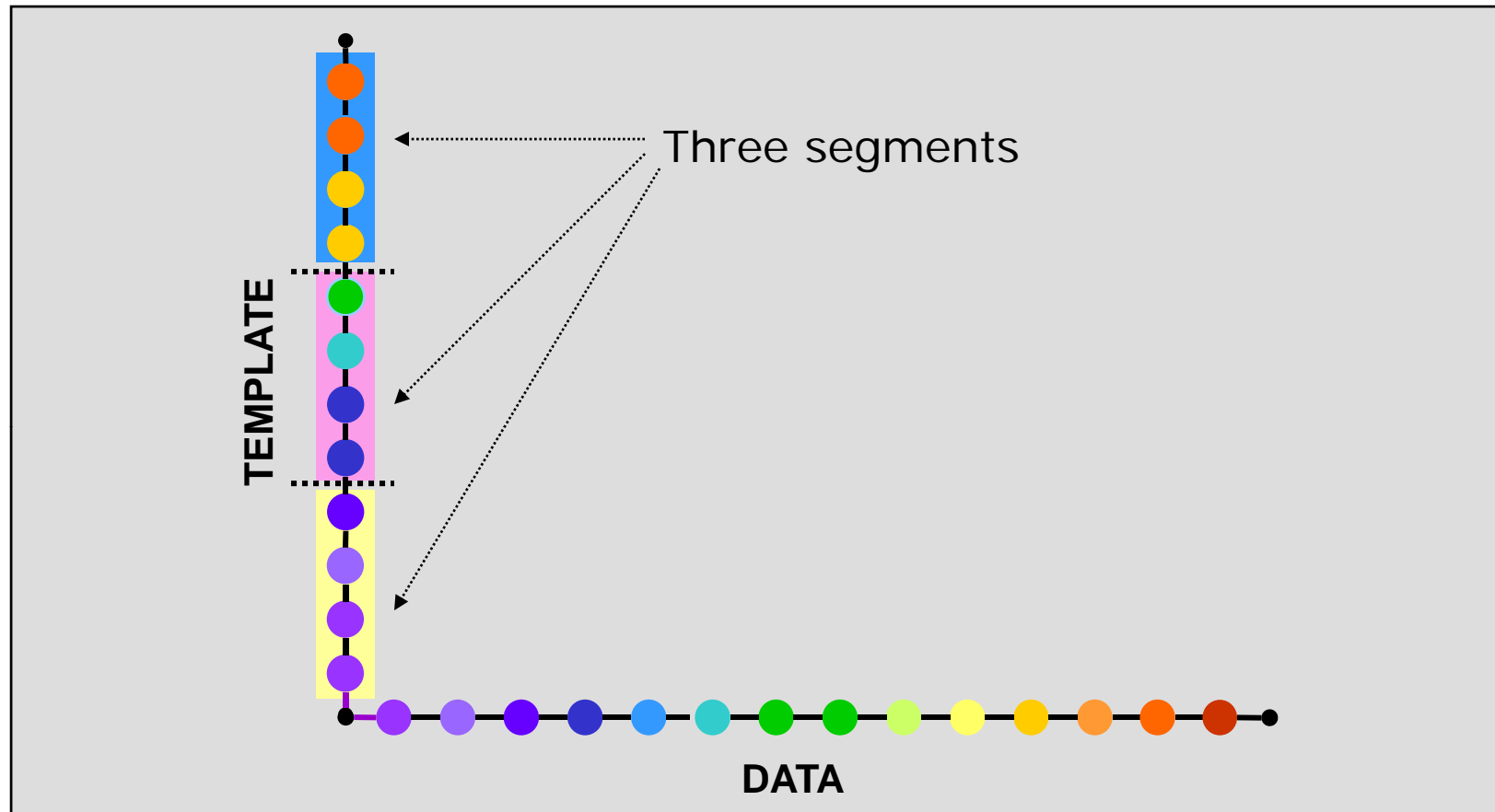
The feature vectors within each segment are assumed to be similar to each other
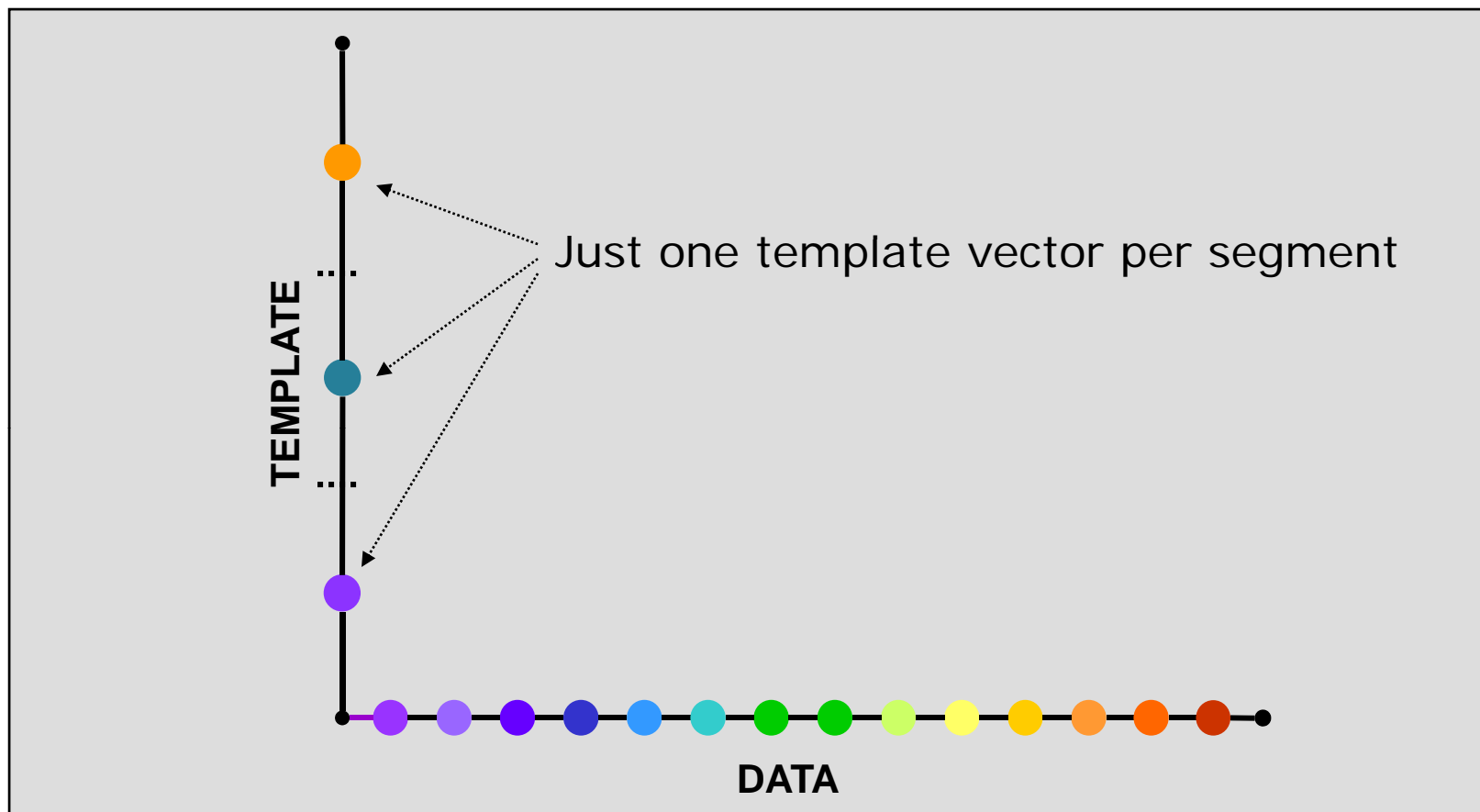
Averaging Each Template Segment

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

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

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

$$m_j = \frac{1}{N_j}\sum_{i\in segment(j)} x(i)$$
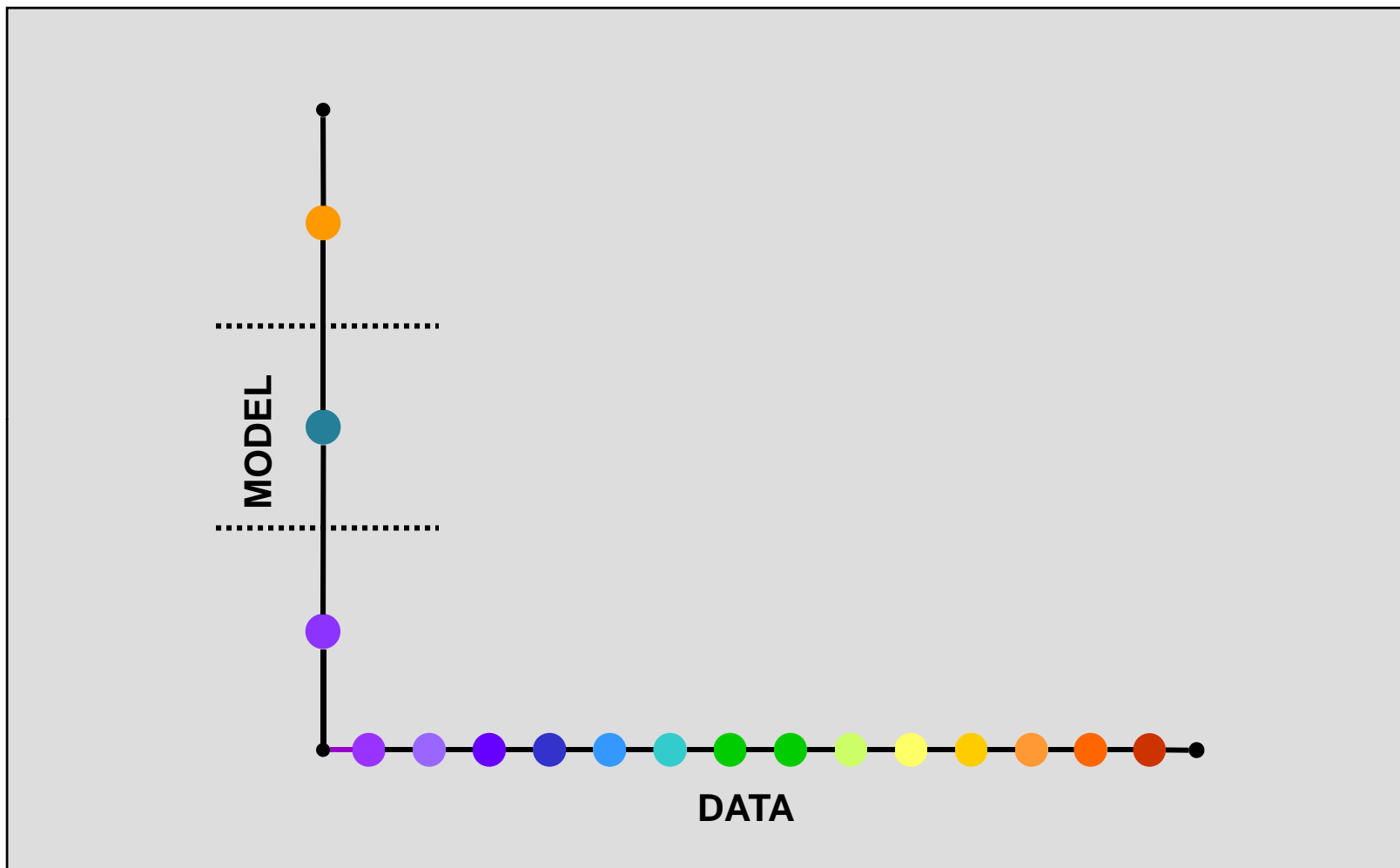
m_j is the model vector for the jth segment

N_j is the number of vectors in the jth segment

$x(i)$ is the ith feature vector

18

Template With One Model Vector Per Segment

Just one template vector per segment
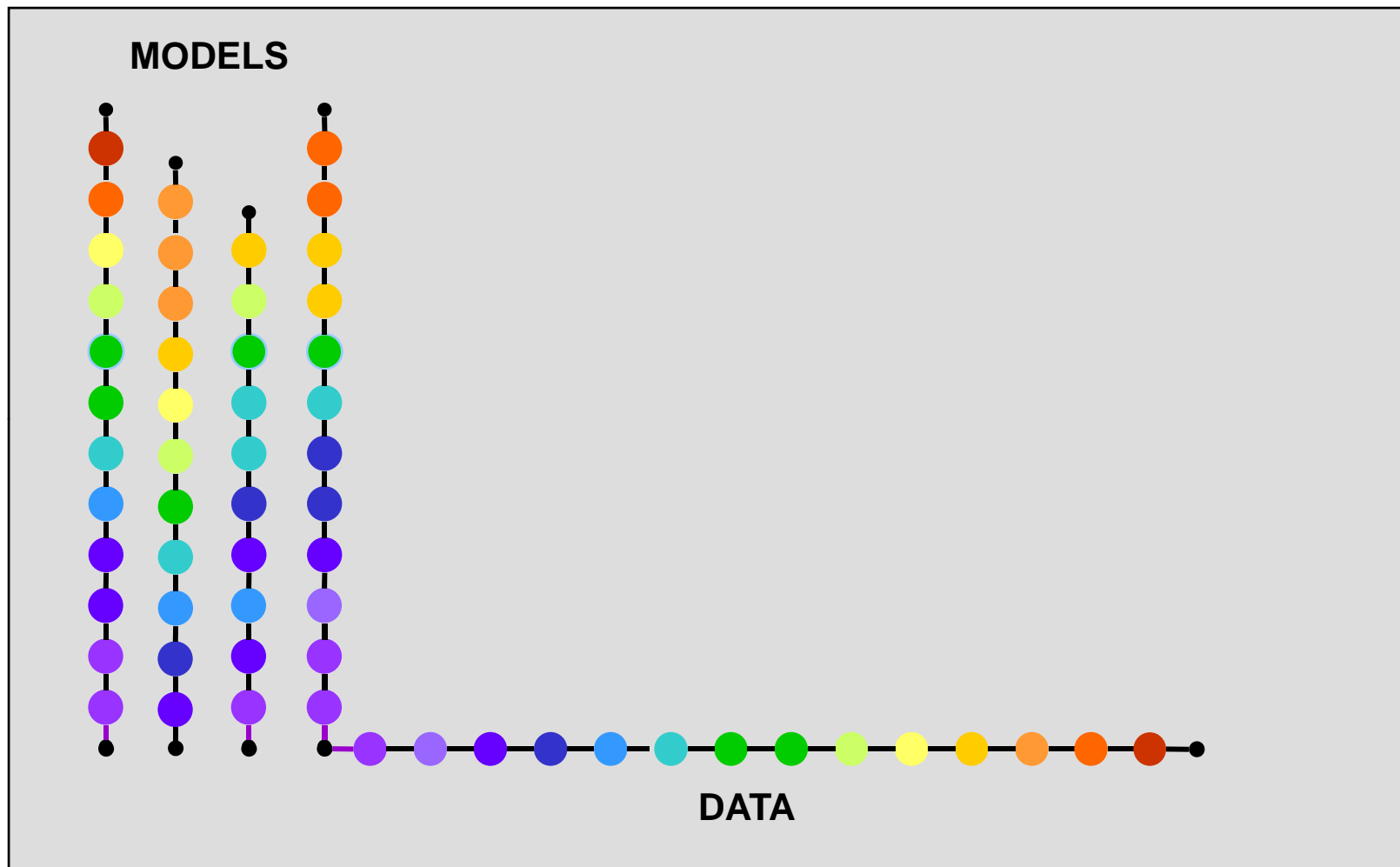
TEMPLATE

DATA

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

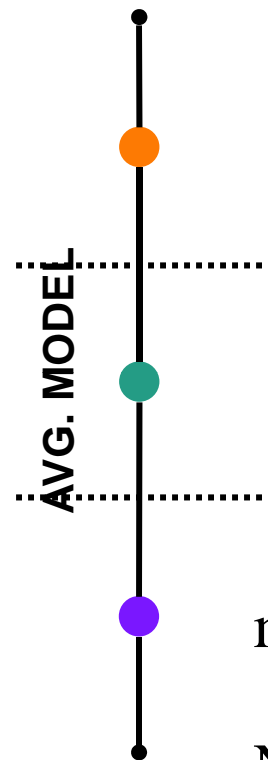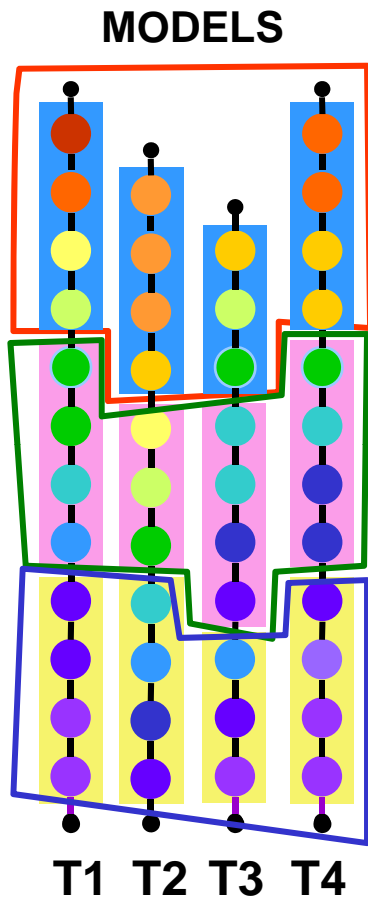# DTW with multiple models

MODELS

DATA

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

MODELS

AVG. MODEL

T1 T2 T3 T4
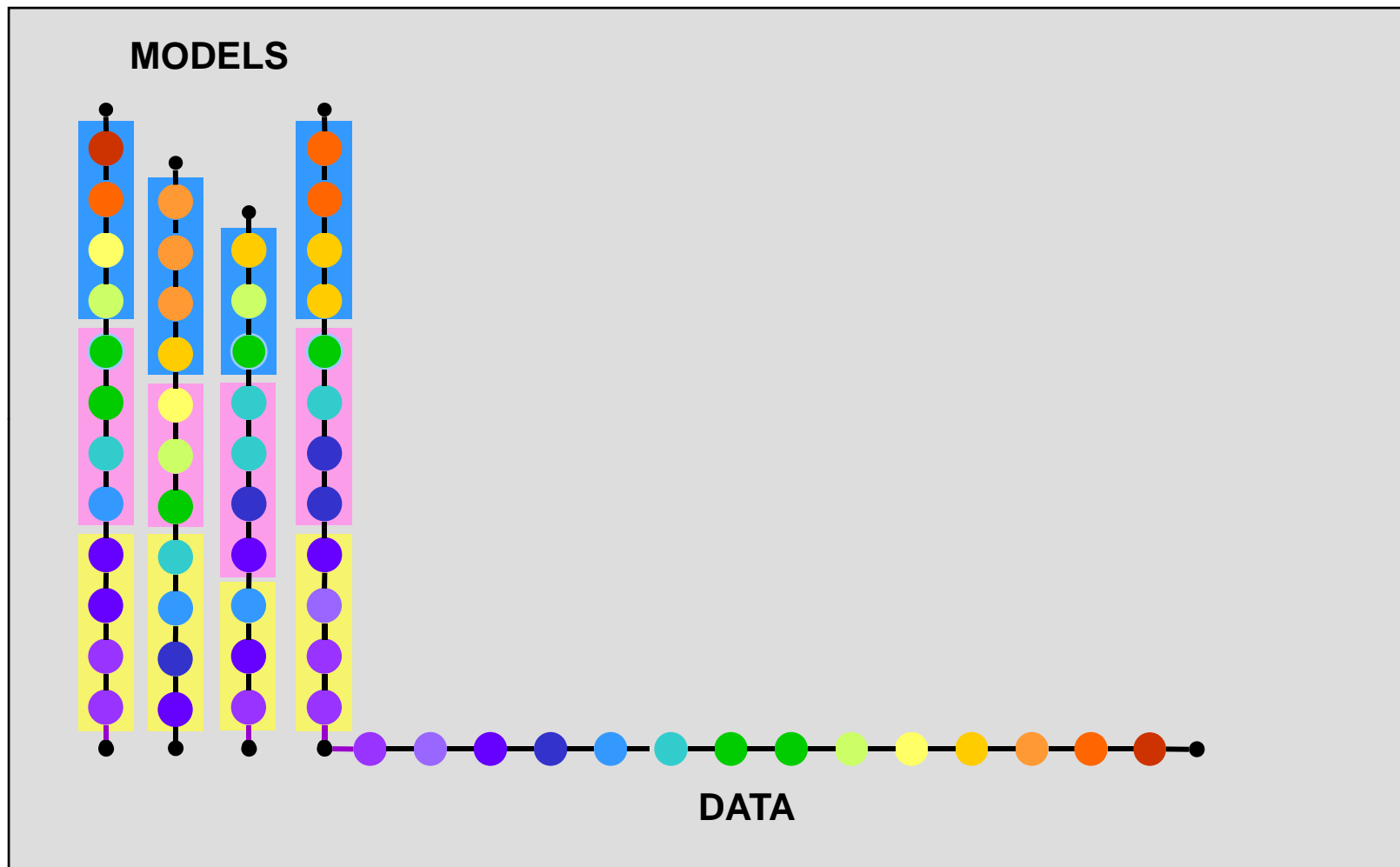
$$m_j = \frac{1}{\sum_k N_{k,j}} \sum_{i \in segment_k(j)} x_k(i)$$
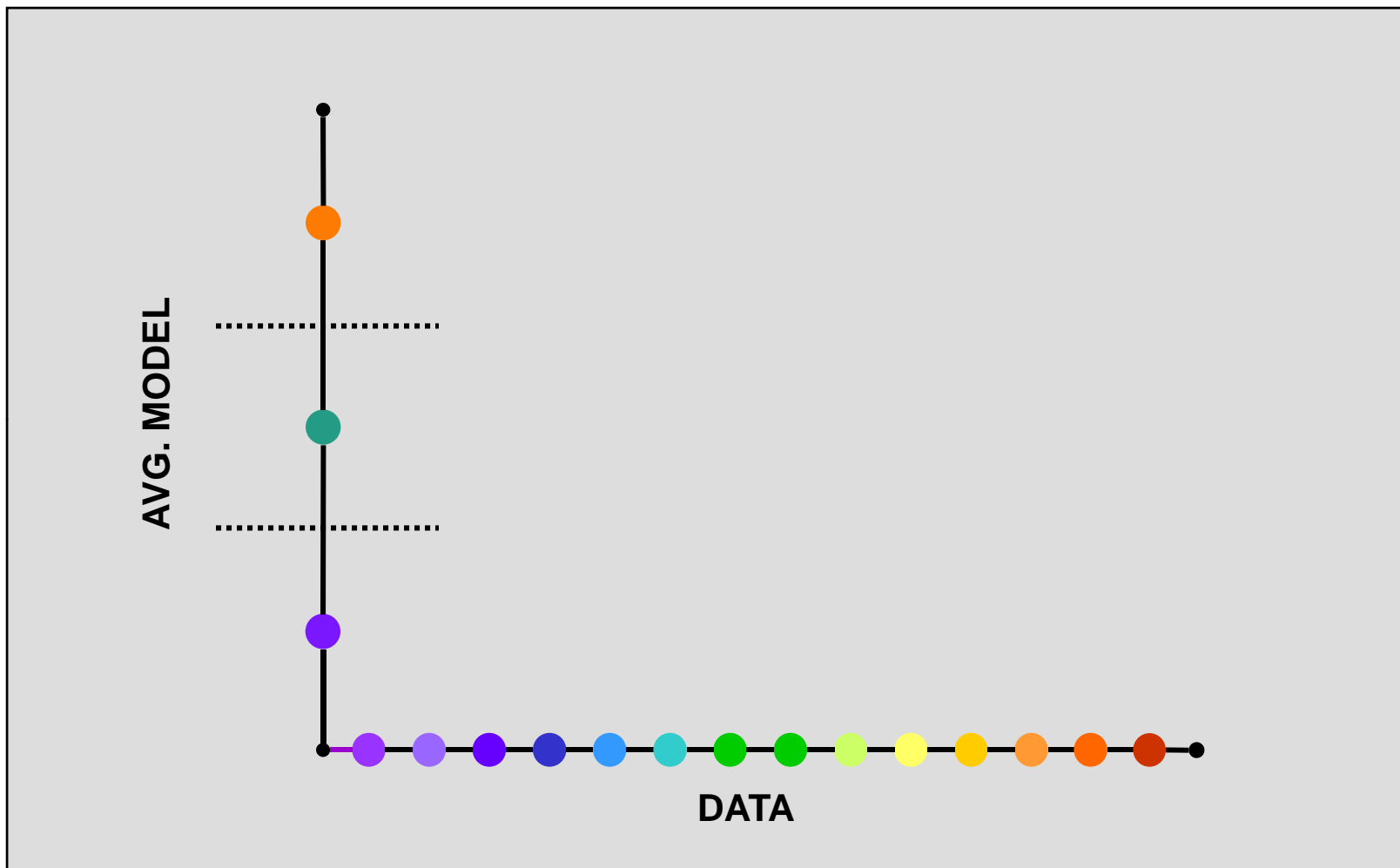
segment$_k$(j) is the jth segment of the kth training sequence

m$_j$ is the model vector for the jth segment

N$_{k,j}$ is the number of training vectors in the jth segment of the kth training sequence

x$_k$(i) is the ith vector of the kth 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

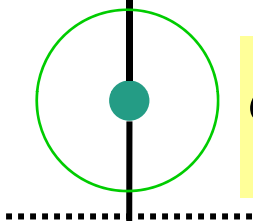# DTW with multiple models

MODELS

T1 T2 T3 T4
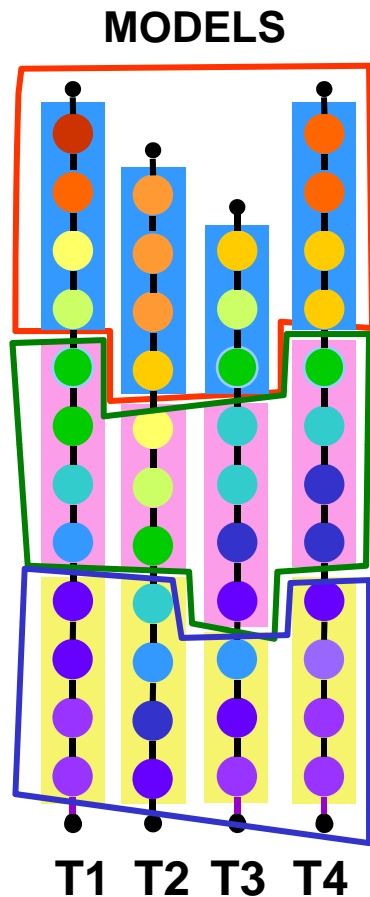
- ## The inherent variation between vectors is different for the different segments
  - E.g. the variation in the colors of the beads in the top segment is greater than that in the bottom segment

- ## 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
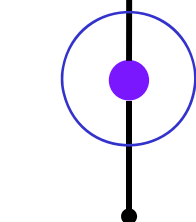
26

DTW with multiple models

MODELS

T1 T2 T3 T4
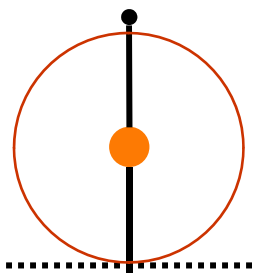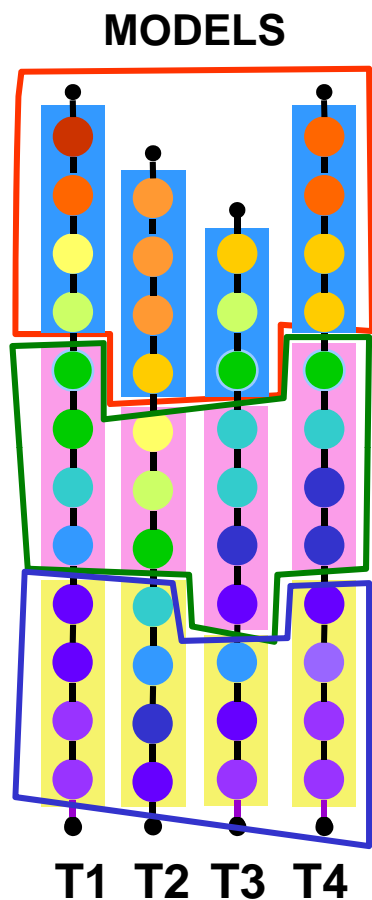
We can define the covariance for each segment using the standard formula for covariance

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

m_j is the model vector for the jth segment

C_j is the covariance of the vectors in the jth segment

27

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

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

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

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

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

- Where the individual variance terms σ^2 are

$$\sigma_{j,l}^2 = \frac{1}{\sum_k N_k} \sum_k \sum_{i \in segment_k(j)} \left(x_{k,l}(i) - m_{j,l} \right)^2$$
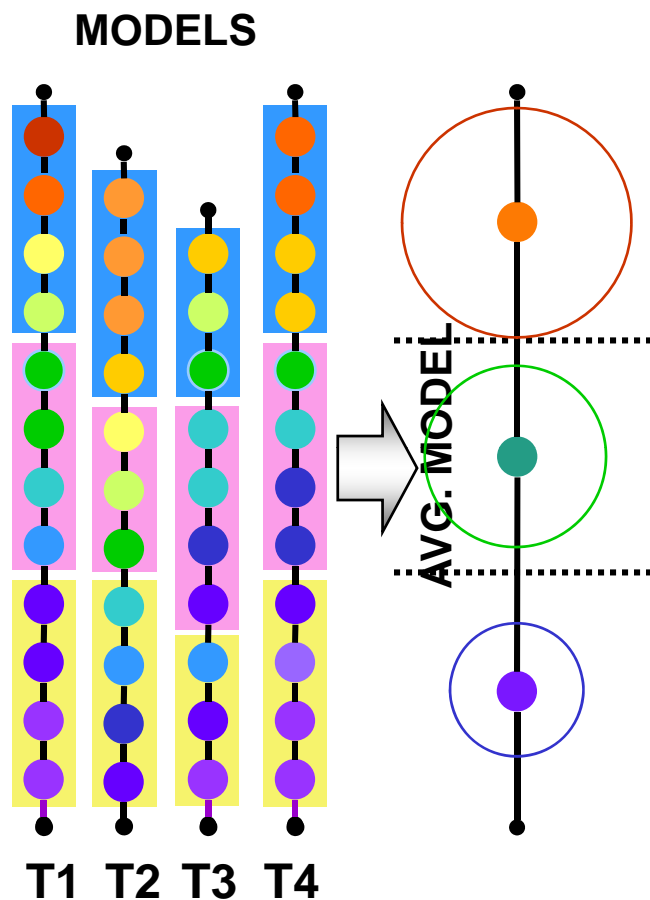
- If we use a negative log Gaussian instead, the modified score (with the diagonal covariance) is

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

29

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

30

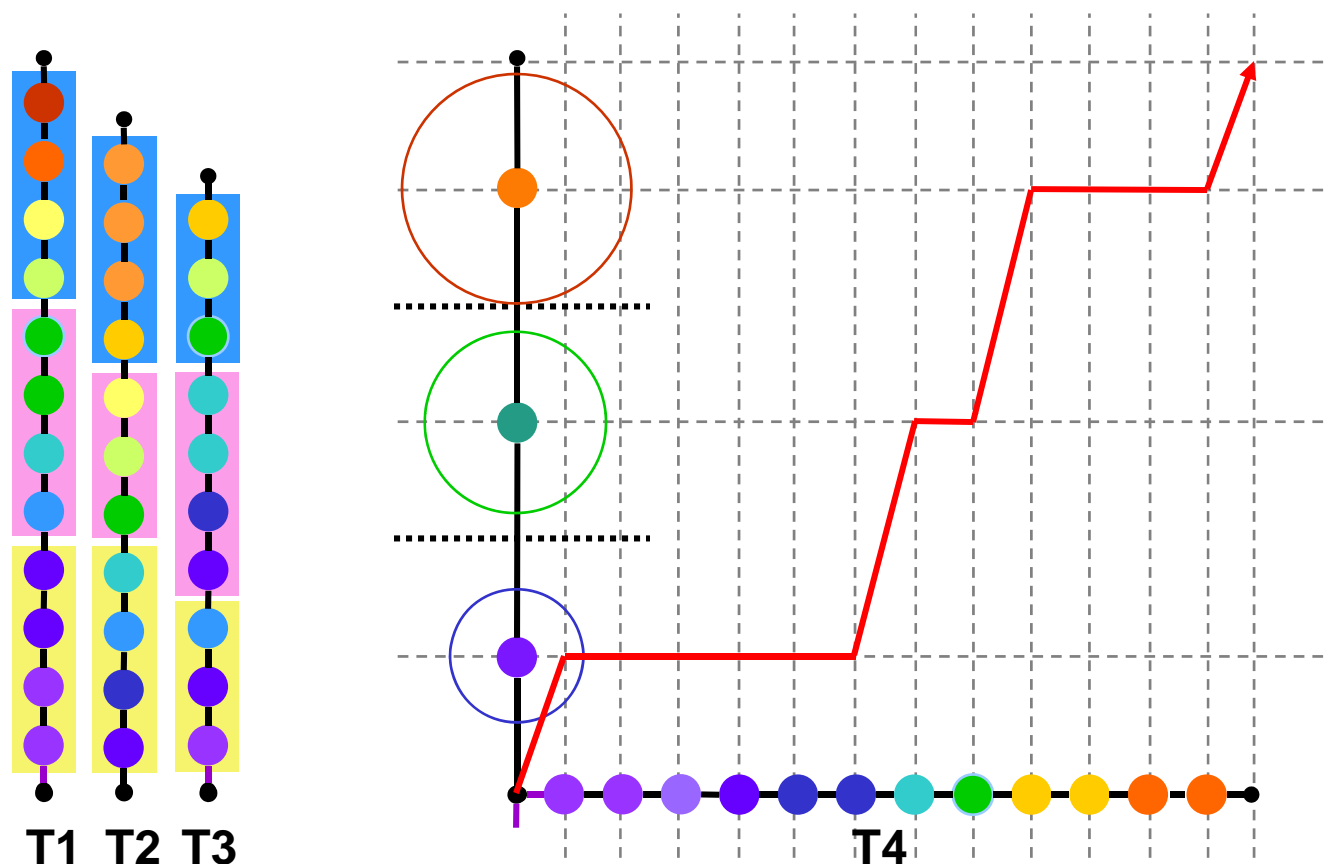# Alignment for training a model from multiple vector sequences

MODELS

AVG. MODEL

T1 T2 T3 T4

Initialize by uniform segmentation

Alignment for training a model from multiple vector sequences

T1 T2 T3

T4
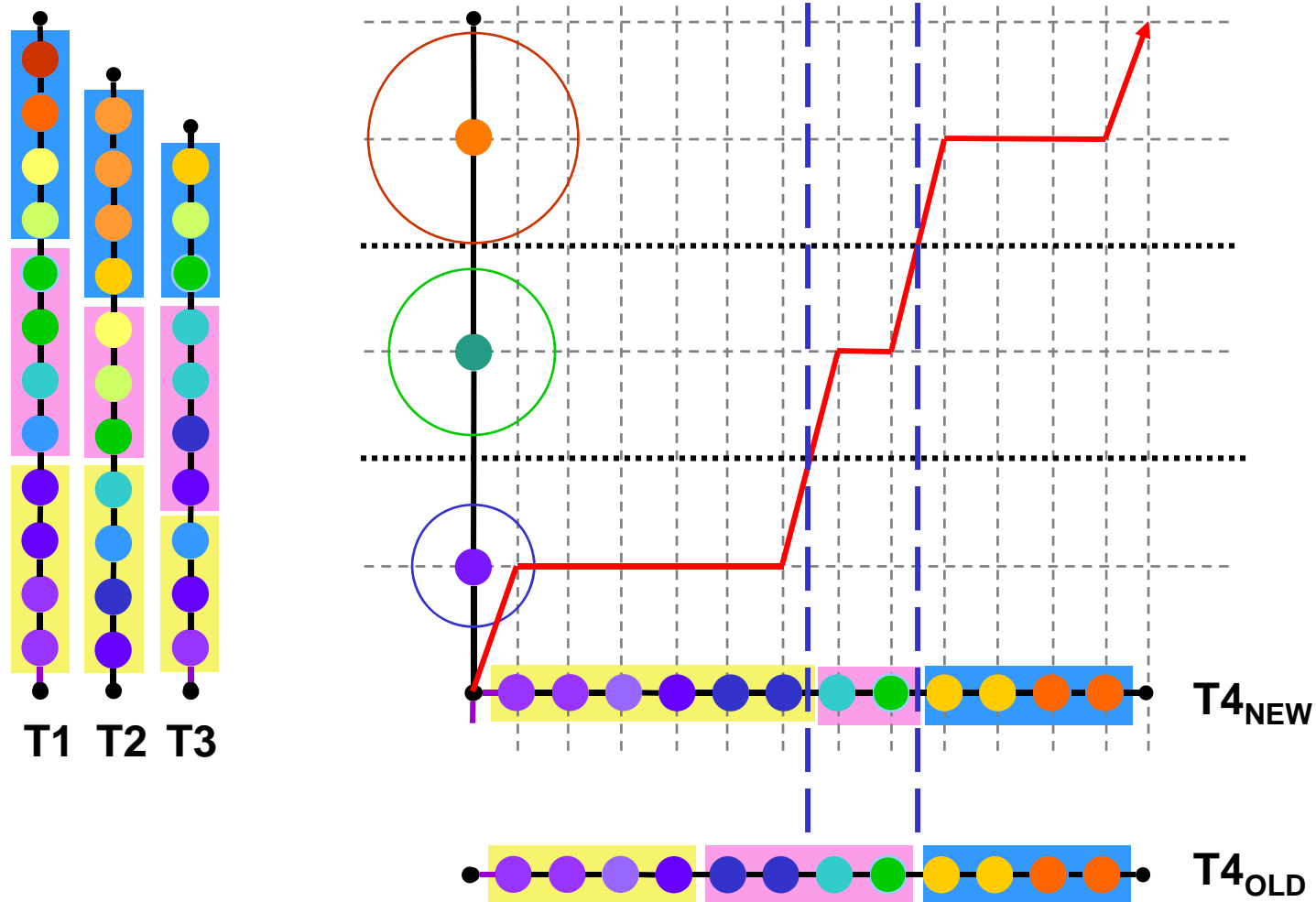
Initialize by uniform segmentation

Alignment for training a model from multiple vector sequences
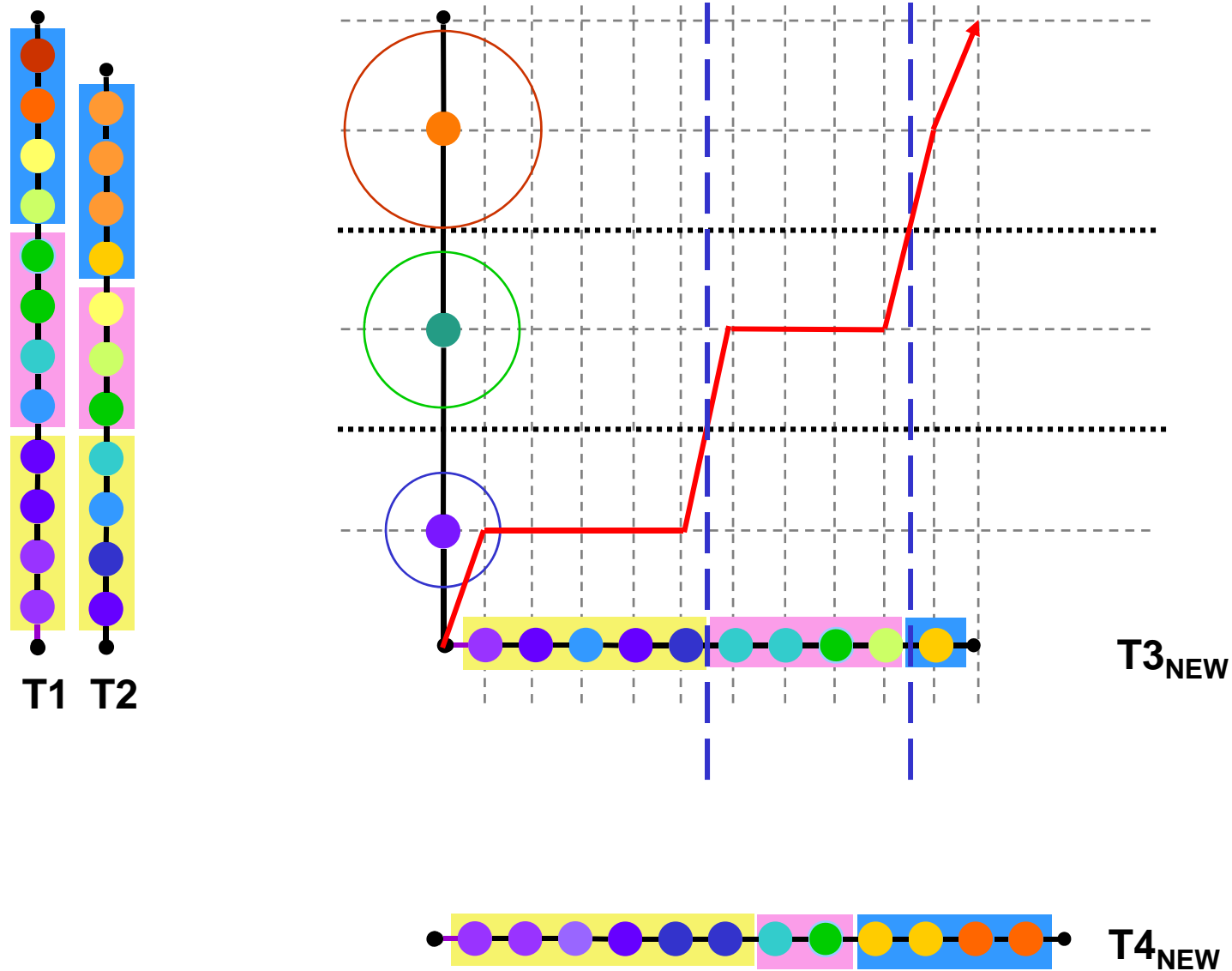

T1 T2 T3

T4

Initialize by uniform segmentation

Align each template to the averaged model to get new segmentations

33

Alignment for training a model from multiple vector sequences

T1   T2   T3

T4$_{NEW}$

T4$_{OLD}$

Alignment for training a model from multiple vector sequences

T1 T2

T3$_{NEW}$

T4$_{NEW}$

Alignment for training a model from multiple vector sequences

T1

T2$_{NEW}$

T3$_{NEW}$

T4$_{NEW}$

36

Alignment for training a model from multiple vector sequences

$T1_{NEW}$

$T2_{NEW}$

$T3_{NEW}$

$T4_{NEW}$

37
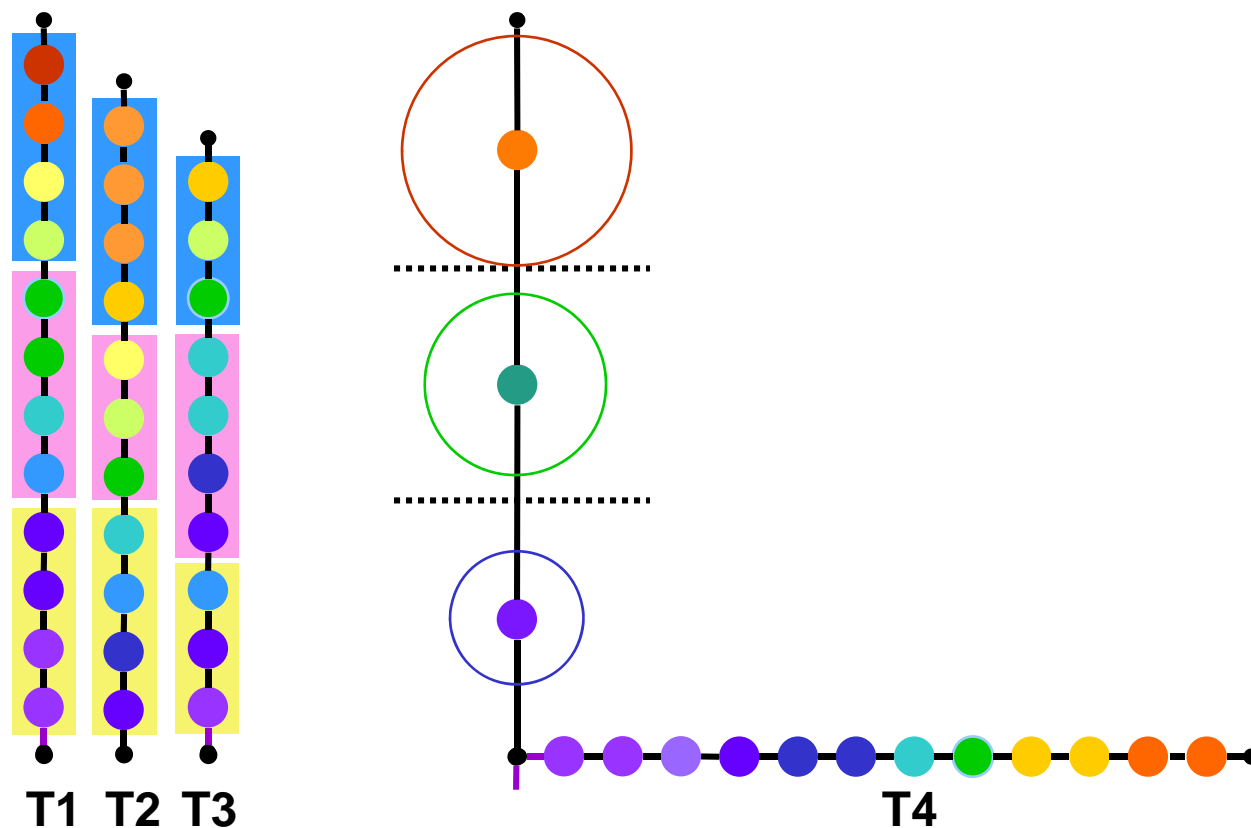
Alignment for training a model from multiple vector sequences

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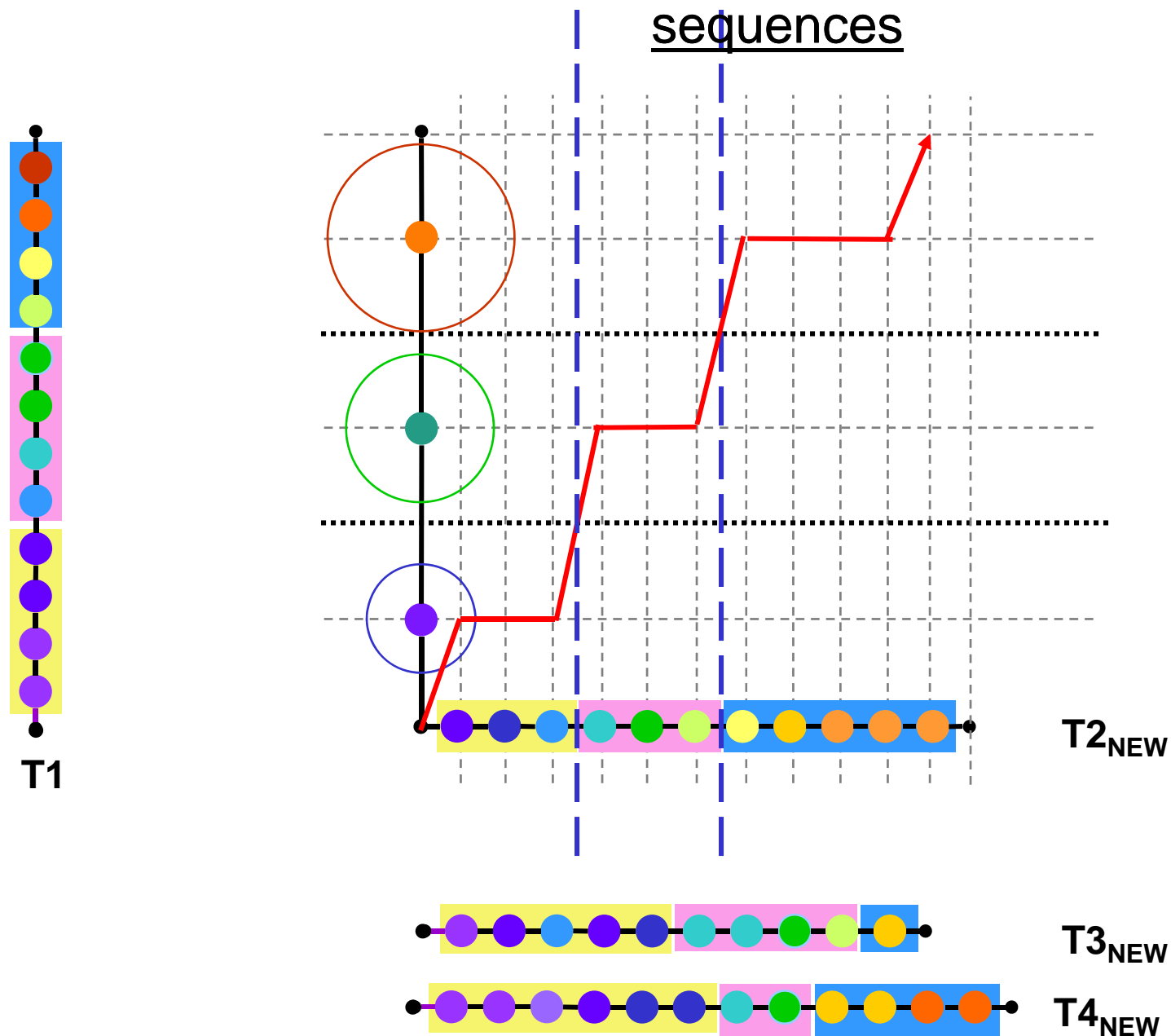Align each template to the averaged model to get new segmentations
Recompute the average model from new segmentations

38

Alignment for training a model from multiple vector sequences

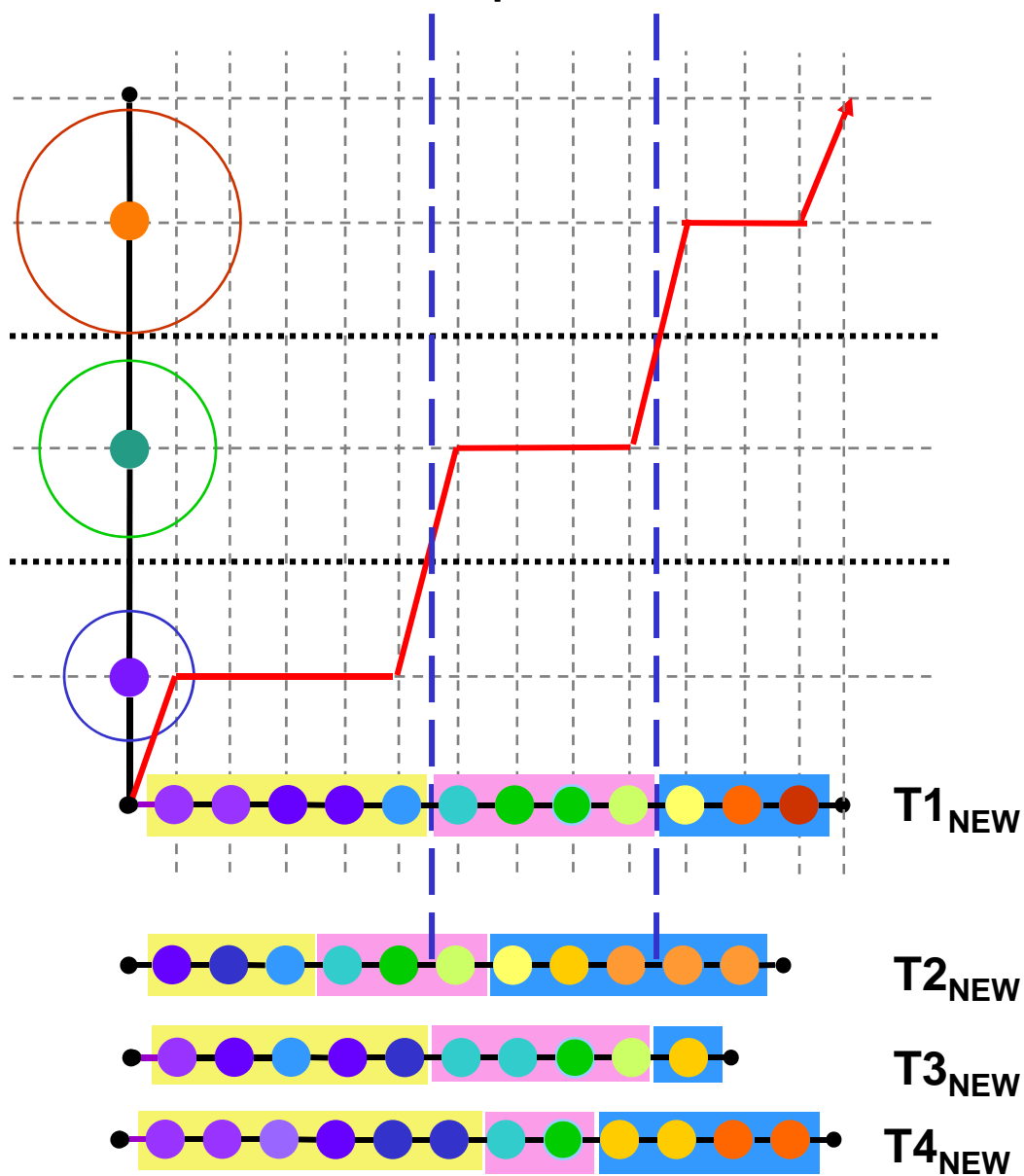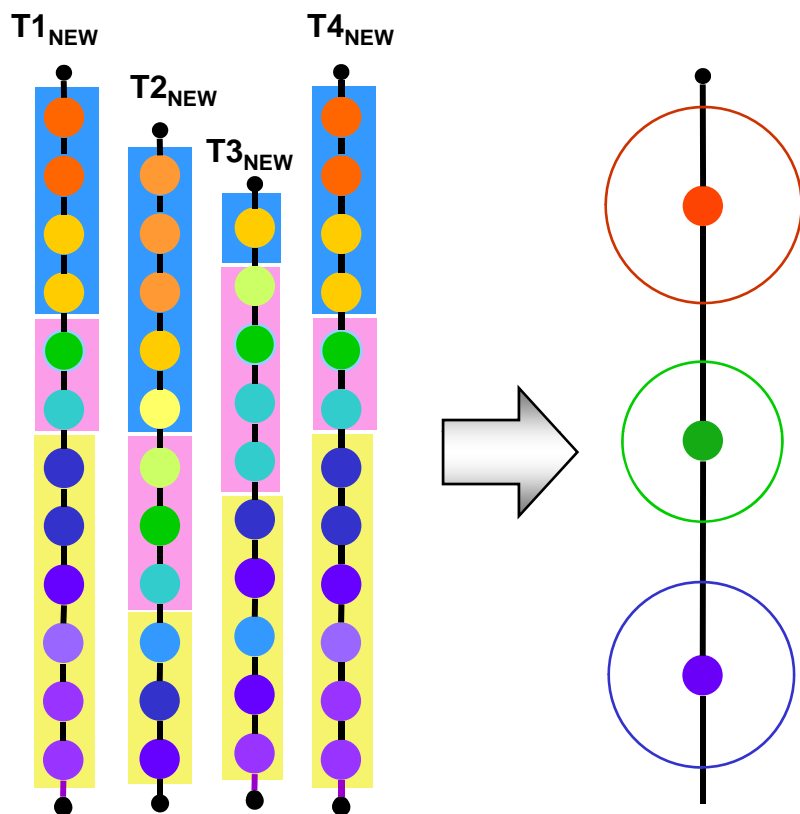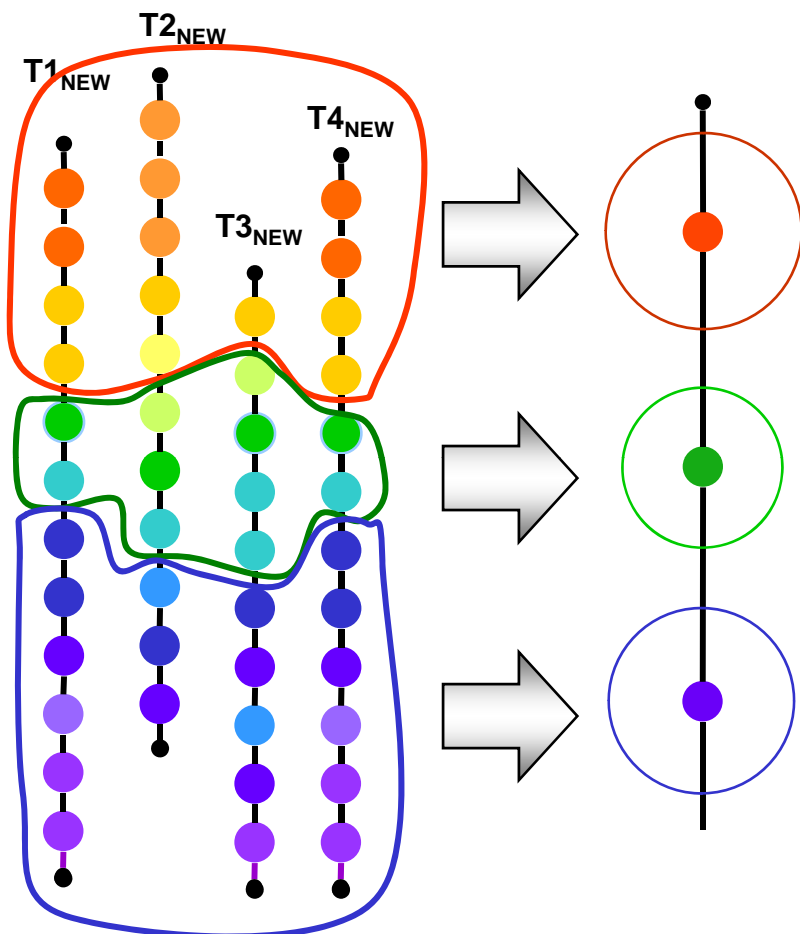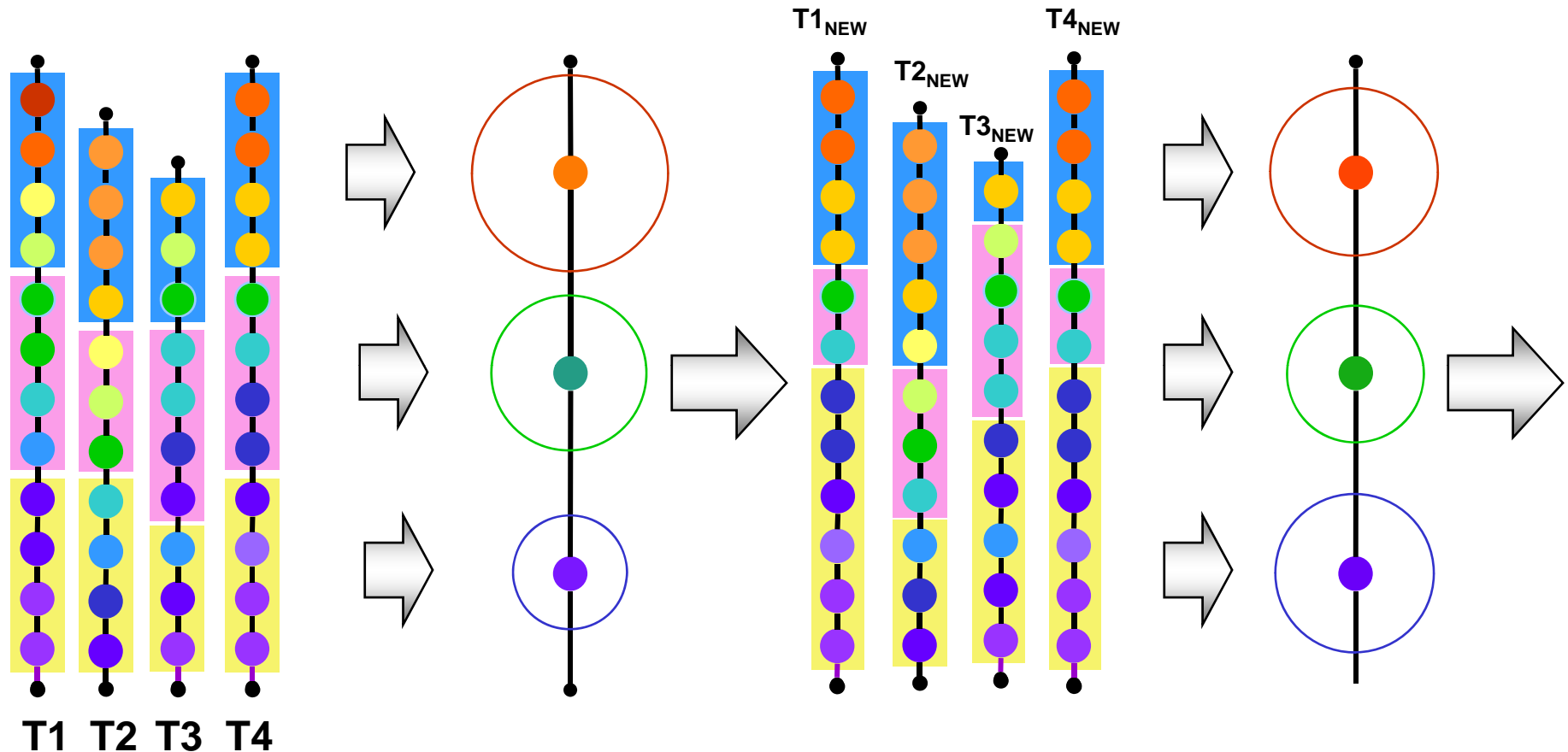# Alignment for training a model from multiple vector sequences

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
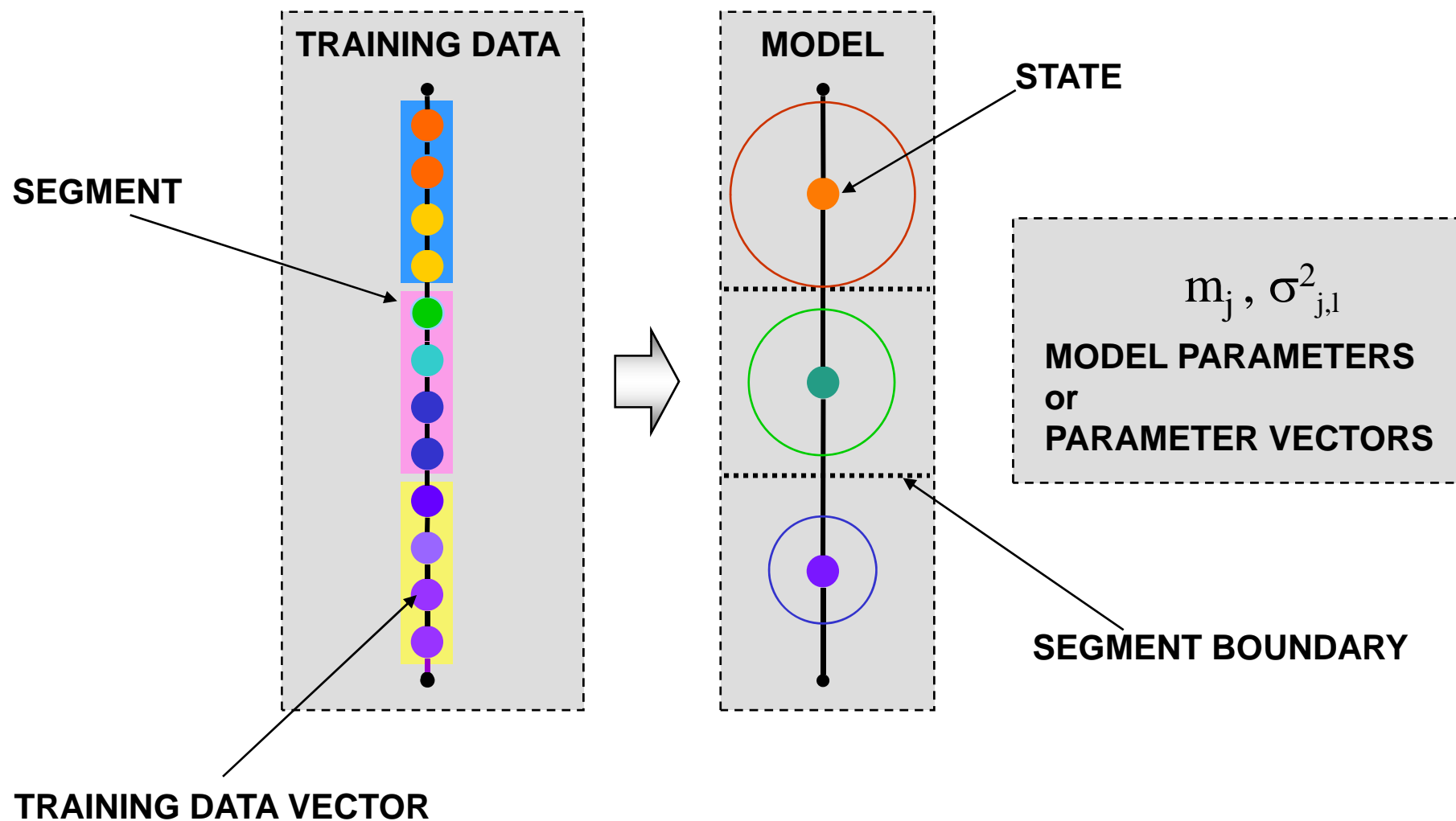
TRAINING DATA

MODEL

STATE

SEGMENT

$$m_j , \sigma^2_{j,l}$$

MODEL PARAMETERS
or
PARAMETER VECTORS

SEGMENT BOUNDARY

TRAINING DATA VECTOR
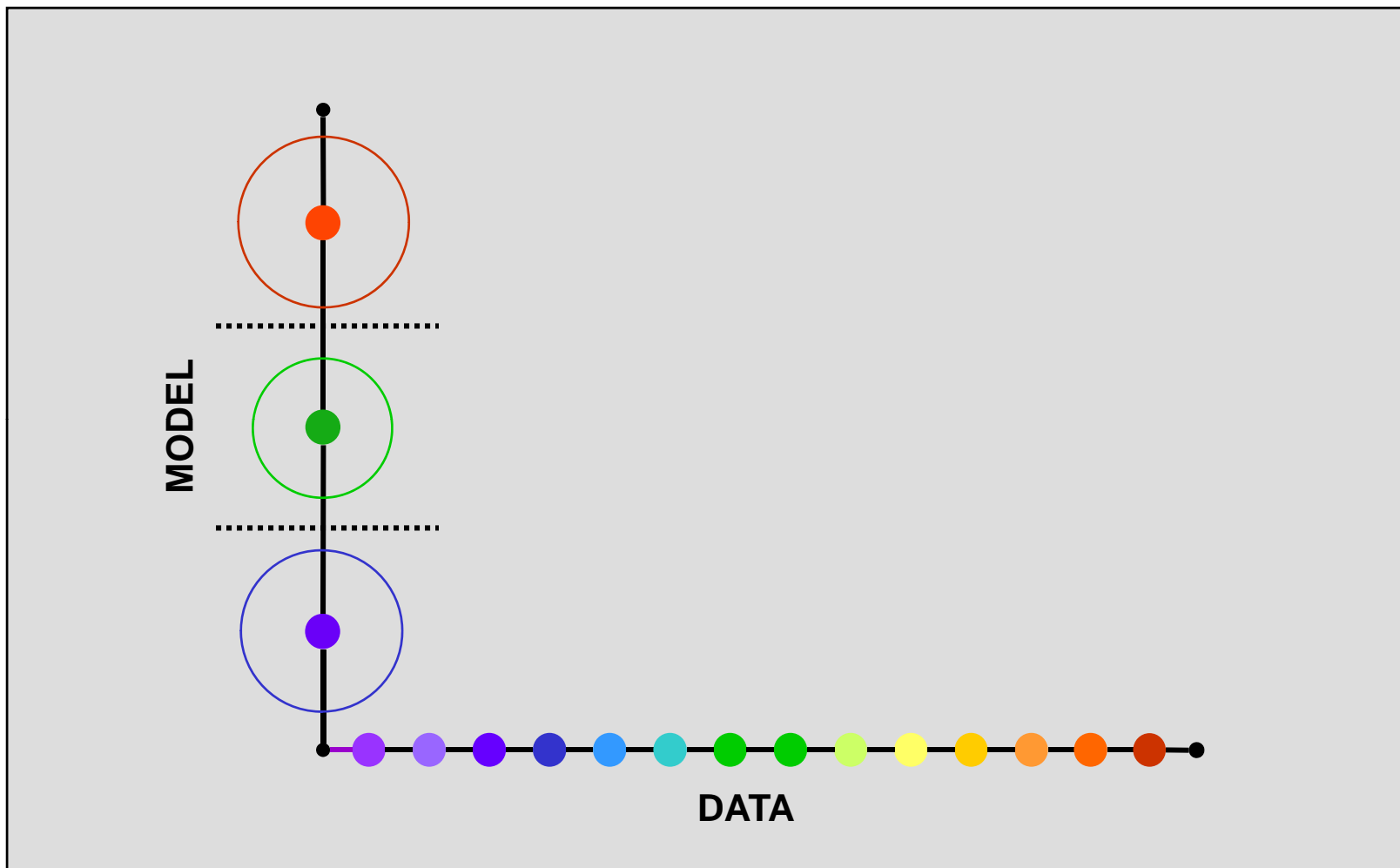
Improving the Templates

☐ Generalization by averaging the templates

☐ Generalization by reducing template length

☐ Accounting for variation within templates represented by the reduced model
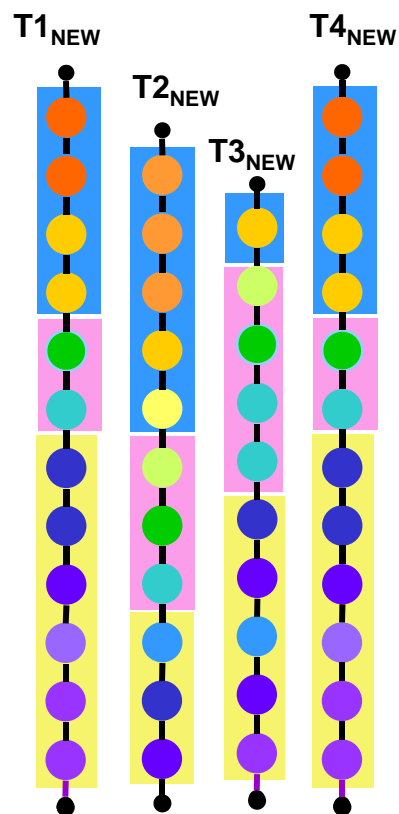
☐ Accounting for varying segment lengths

Transition structures in models

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

Model structure in incomplete.

DTW with multiple models

T1$_{NEW}$

T2$_{NEW}$

T3$_{NEW}$

T4$_{NEW}$

- 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
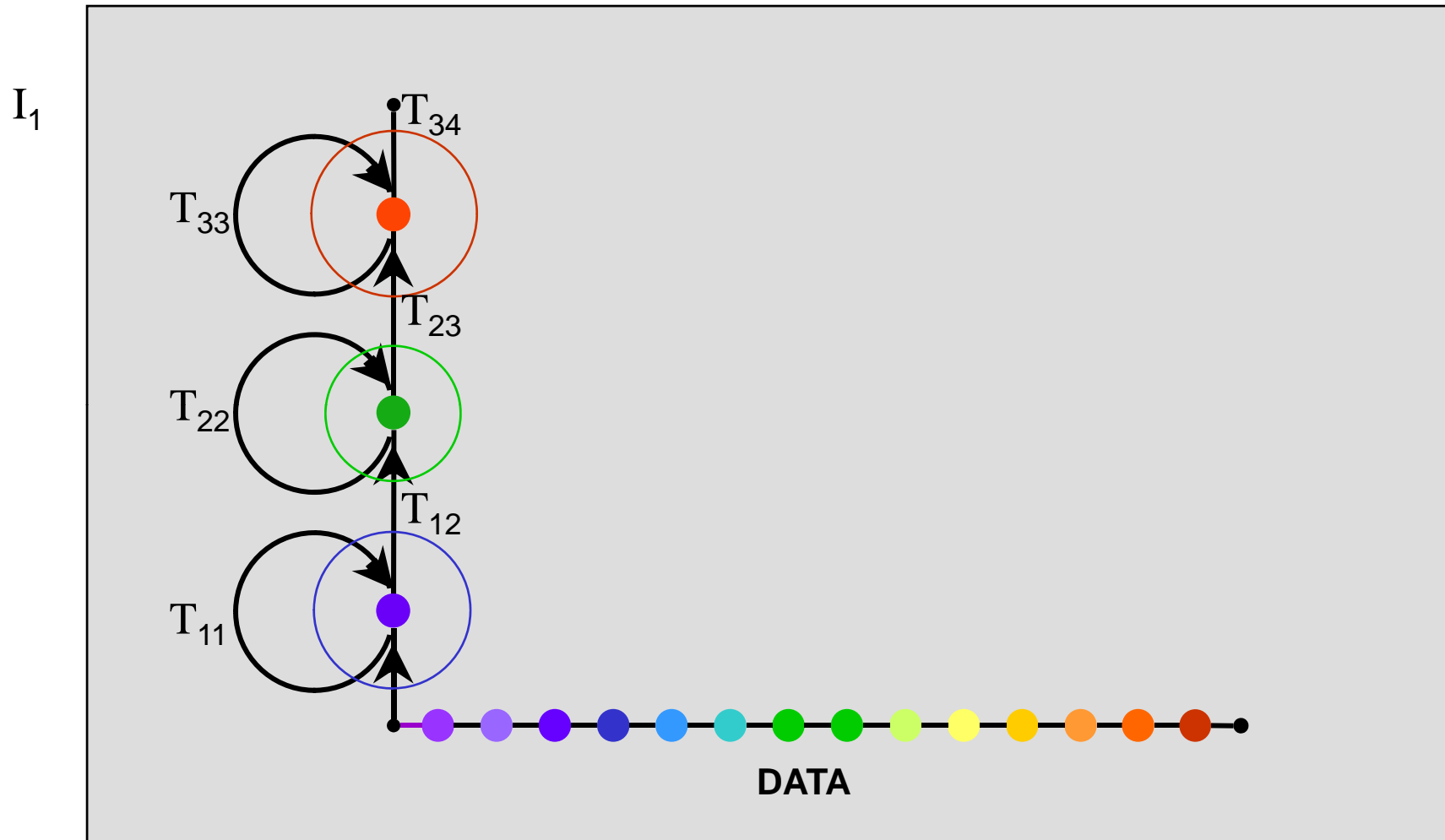
44

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.
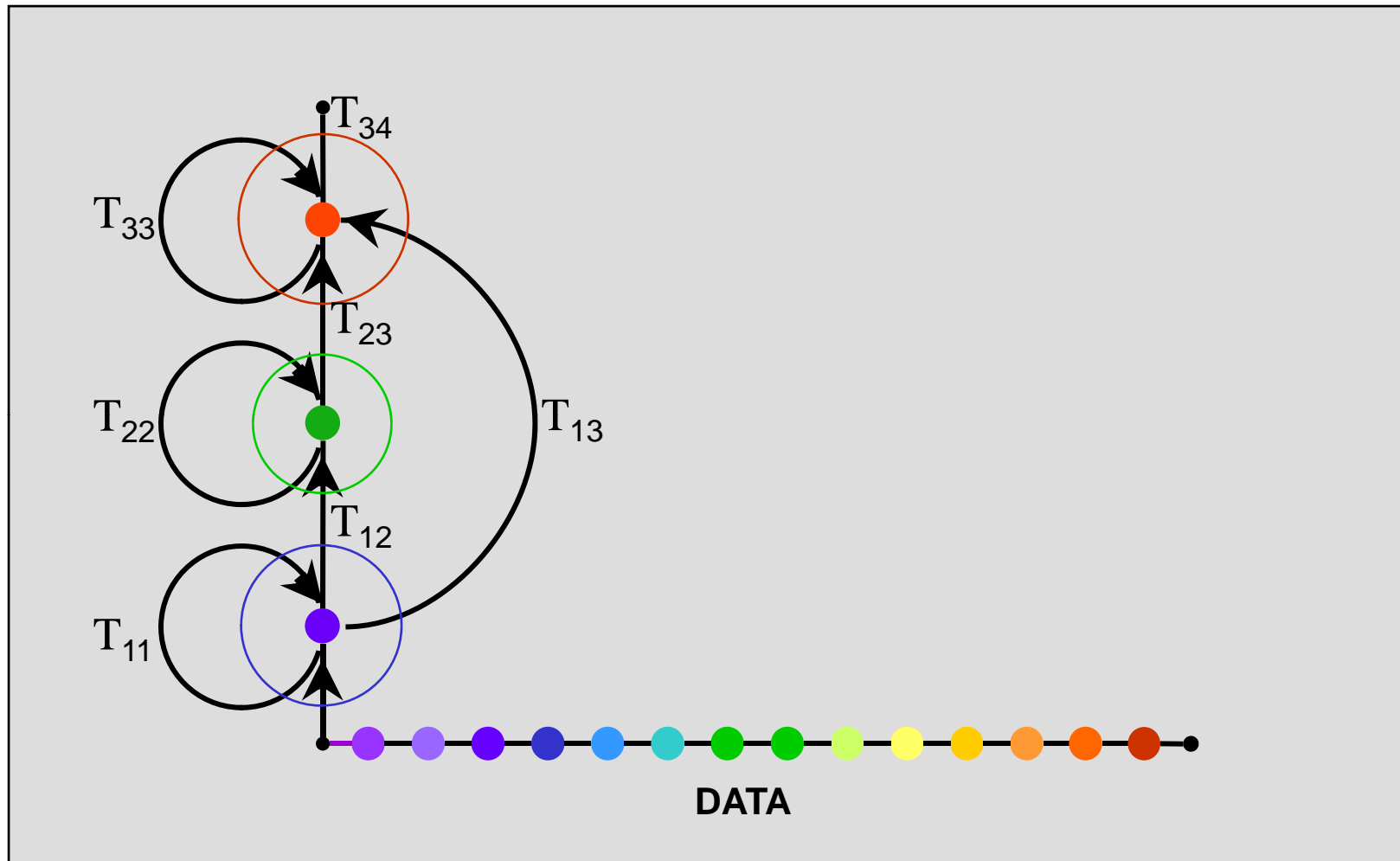
45

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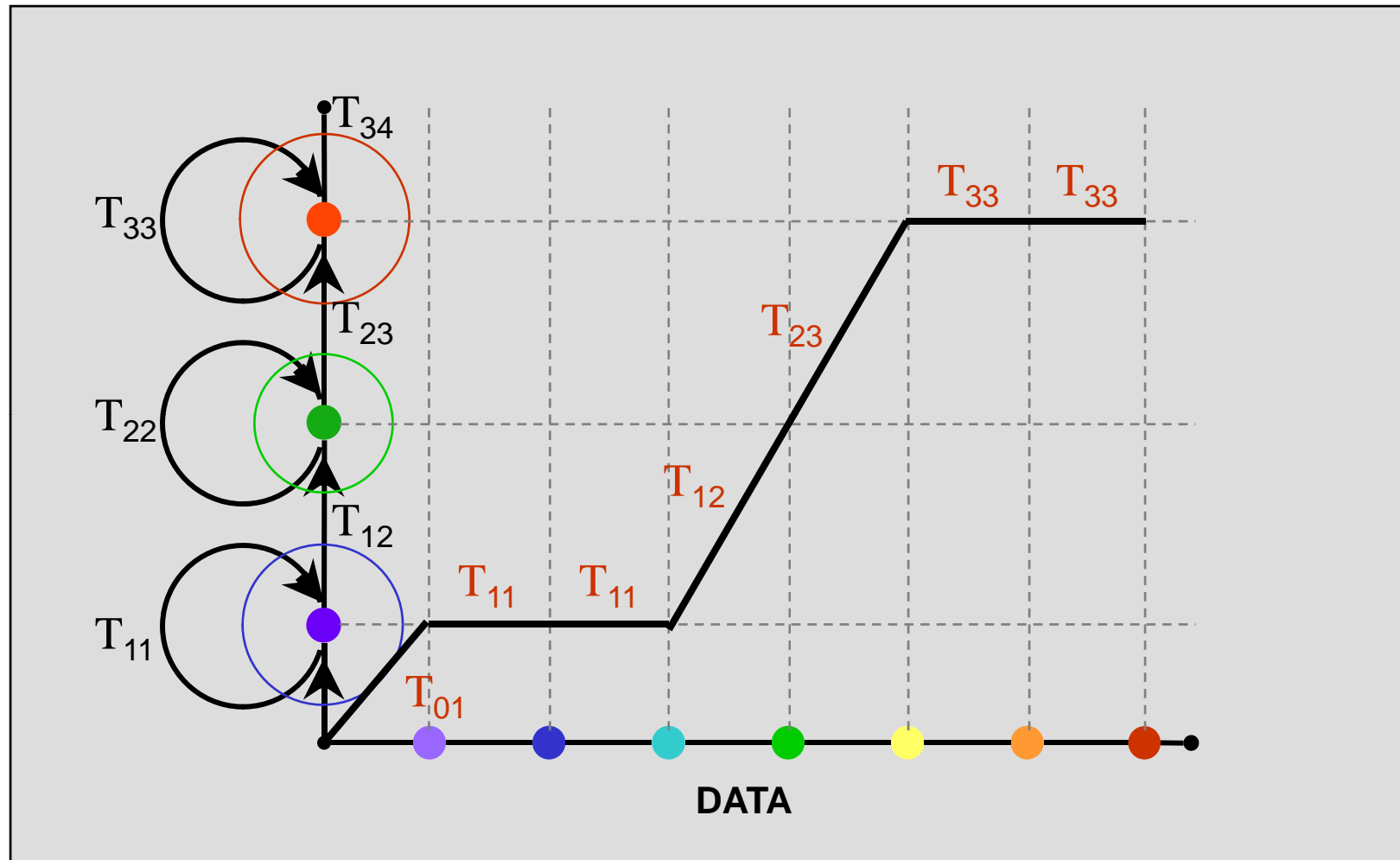
State specific insertion penalties are represented as self transition arcs for model vectors. Horizontal edges within the trellis will incur a penalty associated with the corresponding arc. Every transition within the model can have its own penalty or score

46

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

47

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 ith segment is typically followed by the jth segment, but also rarely by the kth segment, then, if a data vector is within the ith segment, the probability that the next data vector lies in the jth segment is greater than the probability that it lies in the kth 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 ith state, the next data vector will also belong to the ith state

  - T_{ij} is the negative of the log of the probability that if the current data vector belongs to the ith state, the next data vector belongs to the jth state

  - More probable transitions are less penalized. Impossible transitions are 48 infinitely penalized
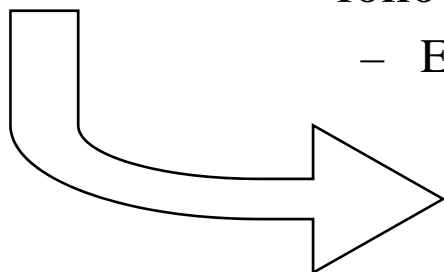
Modified segmental K-means AKA Viterbi training

T1_{NEW} **T4_{NEW}**

T2_{NEW}

T3_{NEW}
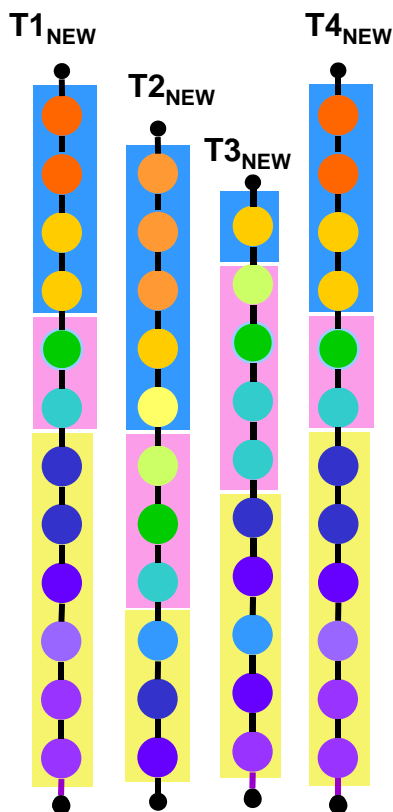
- Transition scores can be easily computed by a simple extension of the segmental K-means algorithm

- 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 ith segment (state) of the kth training sequence

- $N_{k,i,j}$ is the number of vectors in the ith segment (state) of the kth training sequence that were followed by vectors from the jth 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
    $P_{11} = 16/20 = 0.8$; $T_{11} = -\log(0.8)$

49

Modified segmental K-means AKA Viterbi training

T1_{NEW} **T4**_{NEW}

T2_{NEW}

T3_{NEW}

N = 4
N_{01} = 4
N_{02} = 0
N_{03} = 0
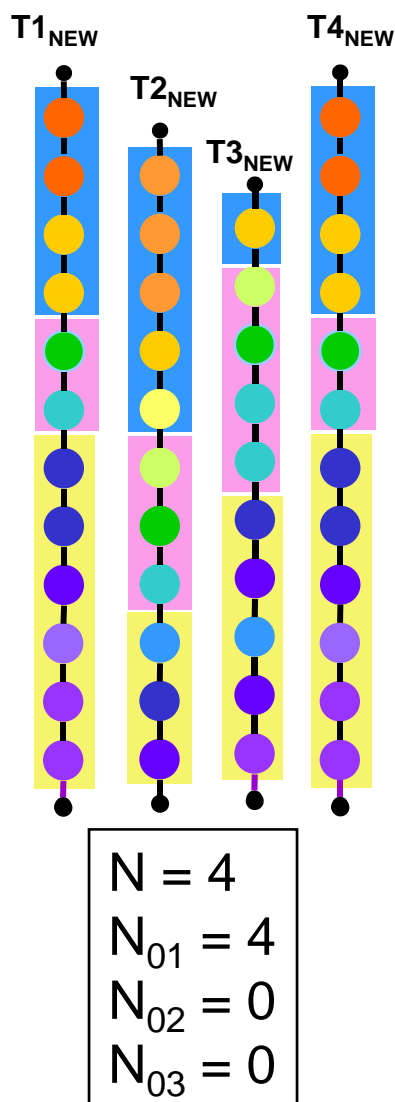
- A special score is the penalty associated with *starting* at a particular state
- In our examples we always begin at the first state
- Enforcing this is equivalent to setting $T_{01} = 0$, $T_{0j} =$ infinity for j != 1
- It is sometimes useful to permit entry directly into later states
  - i.e. permit 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 jth state
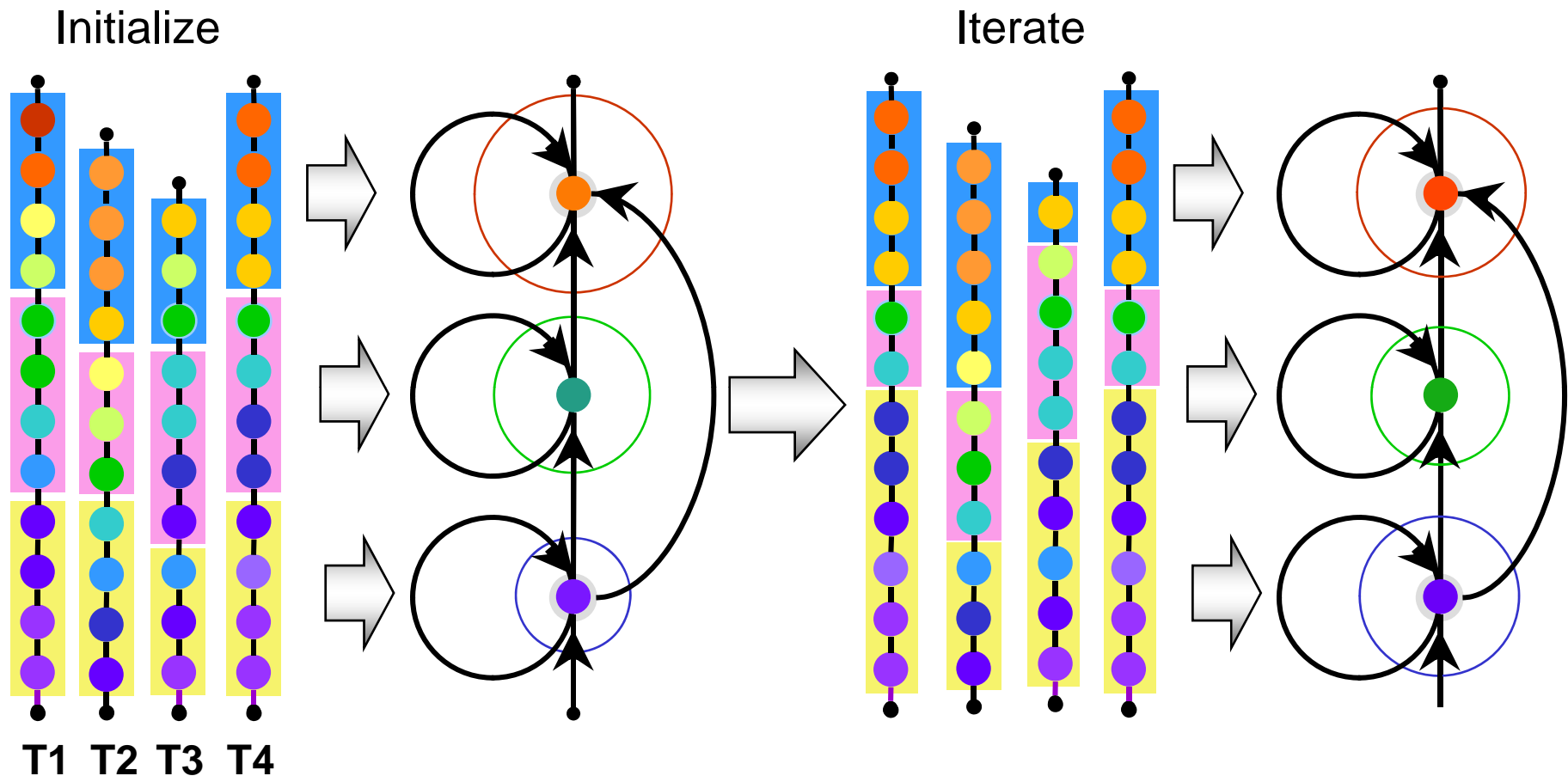
50

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

Initialize

Iterate

T1 T2 T3 T4

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

53