Carnegie Mellon
SCHOOL OF COMPUTER SCIENCE

Dealing with Connected Speech and CI Models

Rita Singh and Bhiksha Raj

Recap and Lookahead

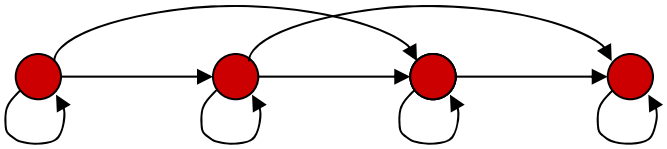
- Covered so far:
  - String-matching-based recognition
    - Learning averaged models
    - Recognition
  - Hidden Markov Models
    - What are HMMs
    - HMM parameter definitions
    - Learning HMMs
    - Recognition of isolated words with HMMs
      - Including how to train HMMs with Gaussian Mixture state output densities
- Continuous speech
  - Isolated-word recognition will only take us so far..
  - Need to deal with strings of words

Connecting Words

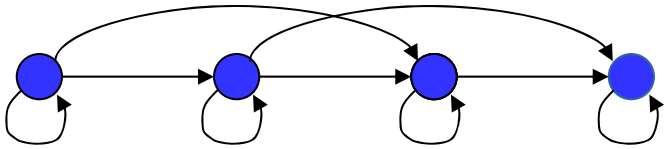
- Most speech recognition applications require word *sequences*
- Even for isolated word systems, it is most convenient to record the training data as sequences of words
  - E.g., if we only need a recognition system that recognizes isolated instances of “Yes” and “No”, it is still convenient to record training data as a word sequences like “Yes No Yes Yes..”
- In all instances the basic unit being modelled is still the *word*
  - Word sequences are formed of words
- Words are represented by HMMs. Models for word sequences are also HMMs composed from the HMMs for words

Composing HMMs for Word Sequences

- Given HMMs for word1 and word2
  - Which are both Bakis topology



word1



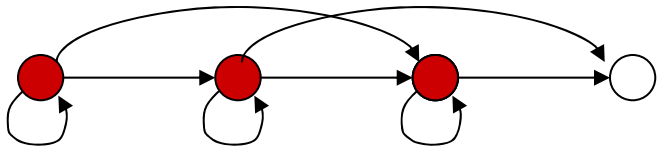
word2

- How do we compose an HMM for the word sequence "word1 word2"
  - Problem: The final state in this model has only a self-transition
  - According the model, once the process arrives at the final state of word1 (for example) it never leaves
  - There is no way to move into the next word

Introducing the Non-emitting state

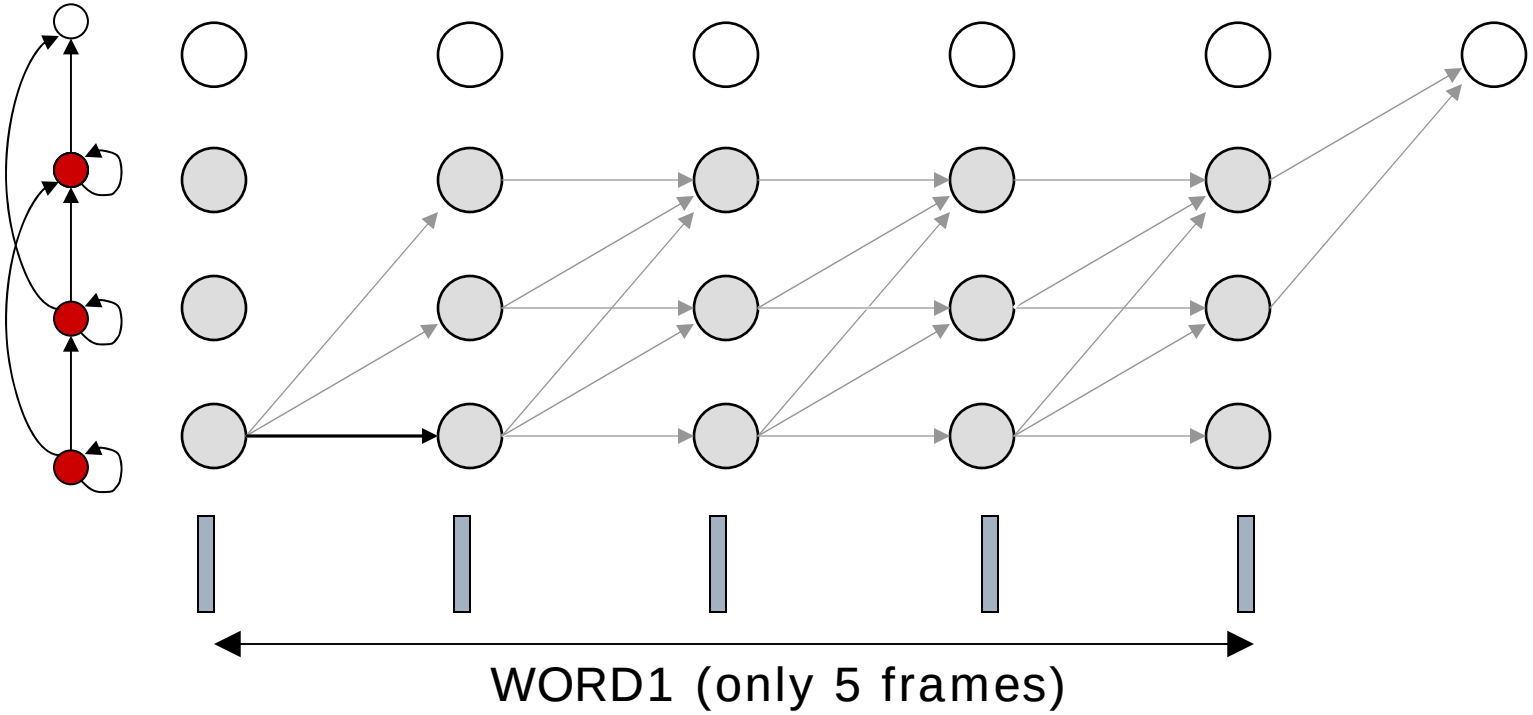
- So far, we have assumed that every HMM state models some output, with some output probability distribution
- Frequently, however, it is useful to include model states that do *not* generate any observation
  - To simplify connectivity

- Such states are called *non-emitting* states or sometimes *null* states
- *NULL STATES CANNOT HAVE SELF TRANSITIONS*
- Example: A word model with a final null state



HMMs with NULL Final State

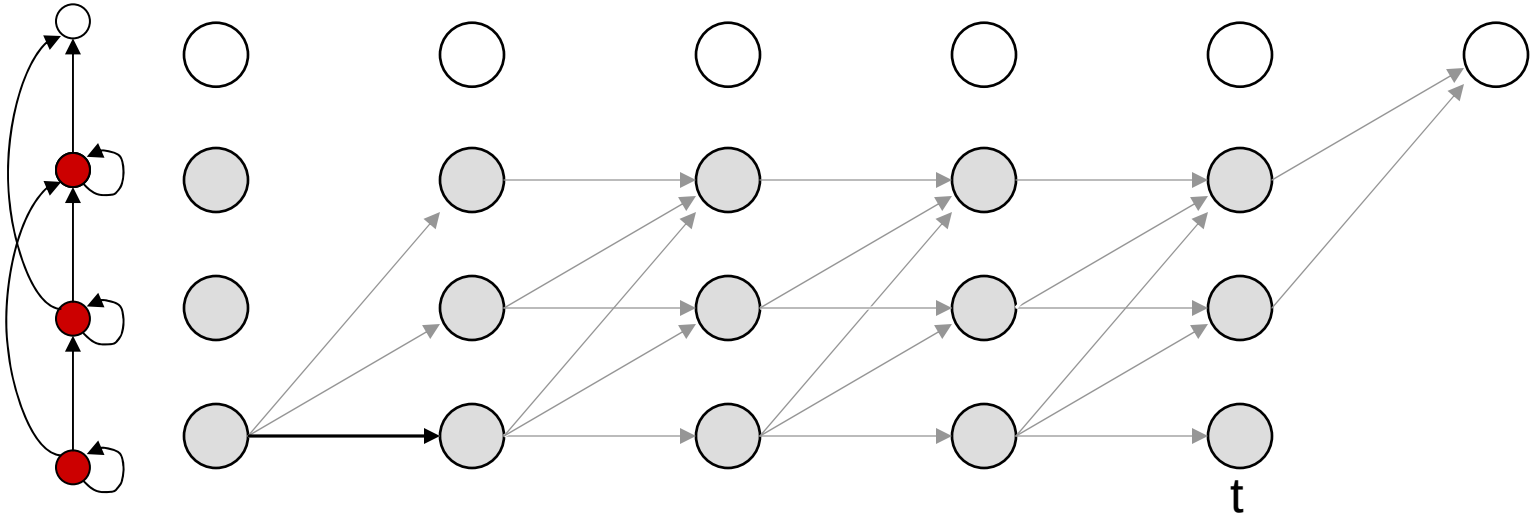
- The final NULL state changes the trellis
  - The NULL state cannot be entered or exited within the word



- If there are exactly 5 vectors in word 5, the NULL state may only be visited after all 5 have been scored

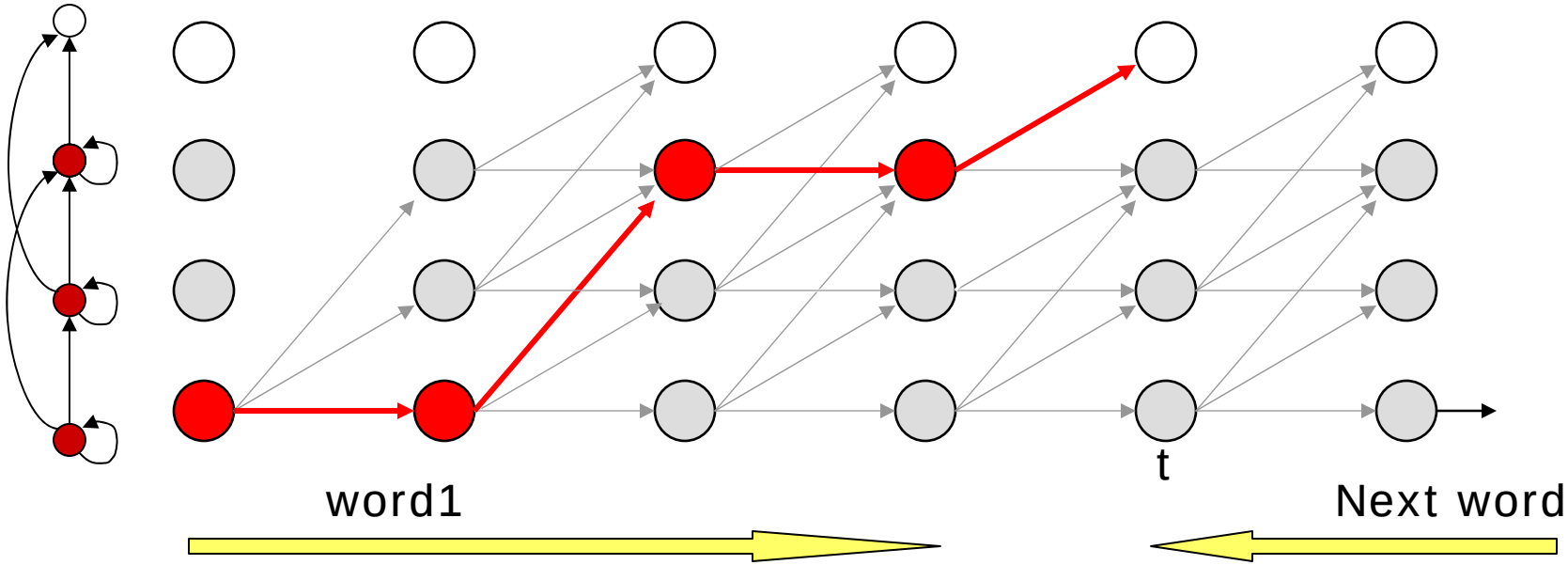
HMMs with NULL Final State

- The final NULL state changes the trellis
  - The NULL state cannot be entered or exited within the word



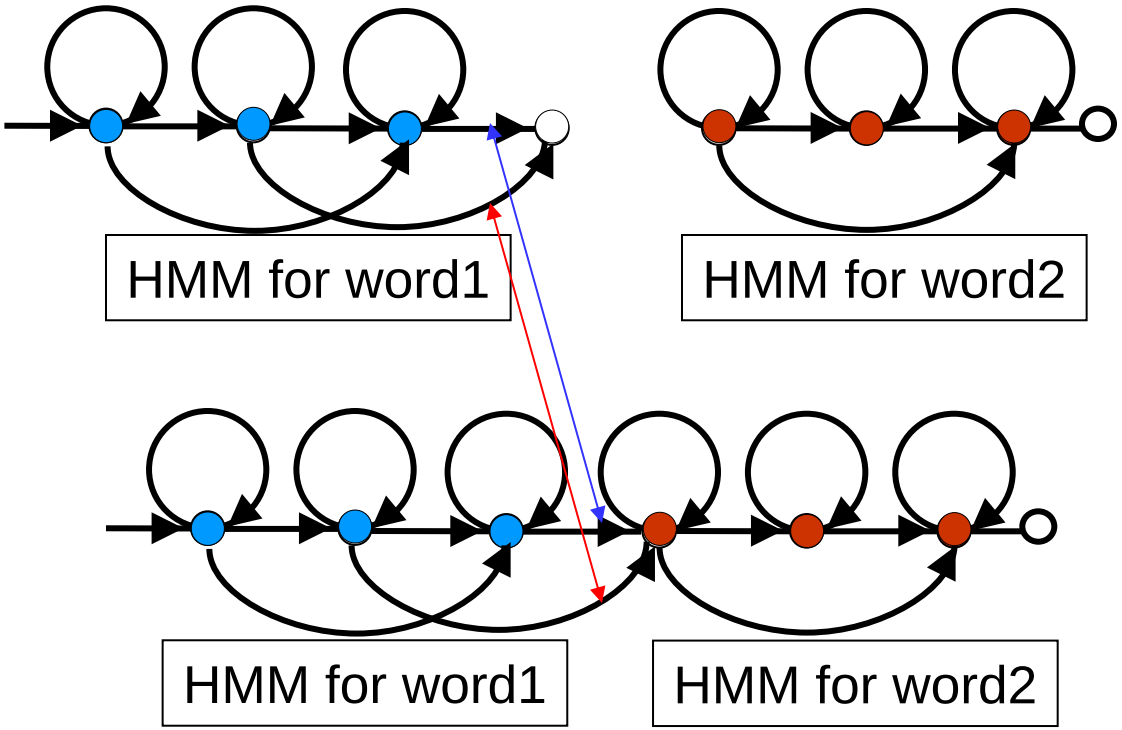
- Standard forward-backward equations apply
  - Except that there is no observation probability P(o|s) associated with this state in the forward pass
  - $\alpha(t+1,3) = \alpha(t,2)\ T_{2,3} + \alpha(t,1)\ T_{1,3}$
  - The backward probability is 1 only for the final state
  - $\beta(t+1,3) = 1.0$; $\beta(t+1,s) = 0$ for $s = 0,1,2$

The NULL final state



- The probability of transitioning into the NULL final state at any time *t* is the probability that the observation sequence for the word will end at time *t*
- Alternately, it represents the probability that the observation will exit the word at time *t*

Connecting Words with Final NULL States

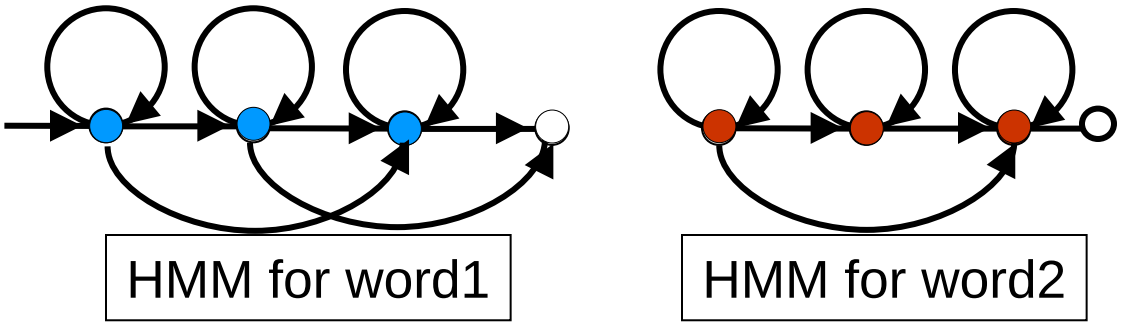


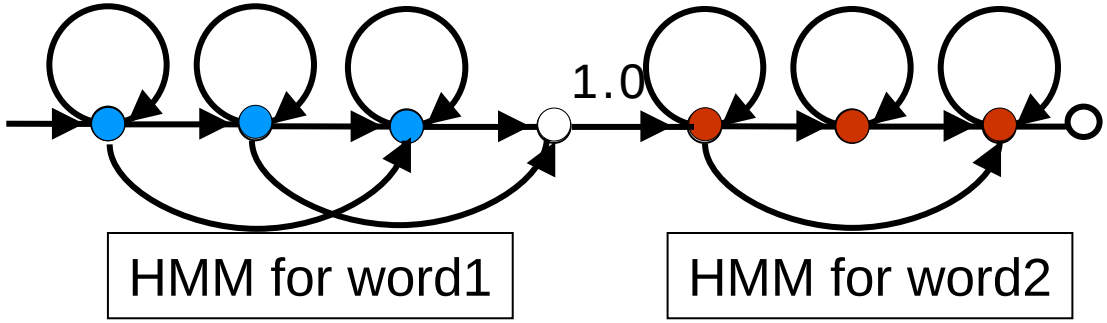
- The probability of leaving word 1 (i.e the probability of going to the NULL state) is the same as the probability of entering word2
  - The transitions pointed to by the two ends of each of the colored arrows are the same

Retaining a Non-emitting state between words

- In some cases it may be useful to retain the non-emitting state as a connecting state
  - The probability of entering word 2 from the non-emitting state is 1.0
  - This is the only transition allowed from the non-emitting state

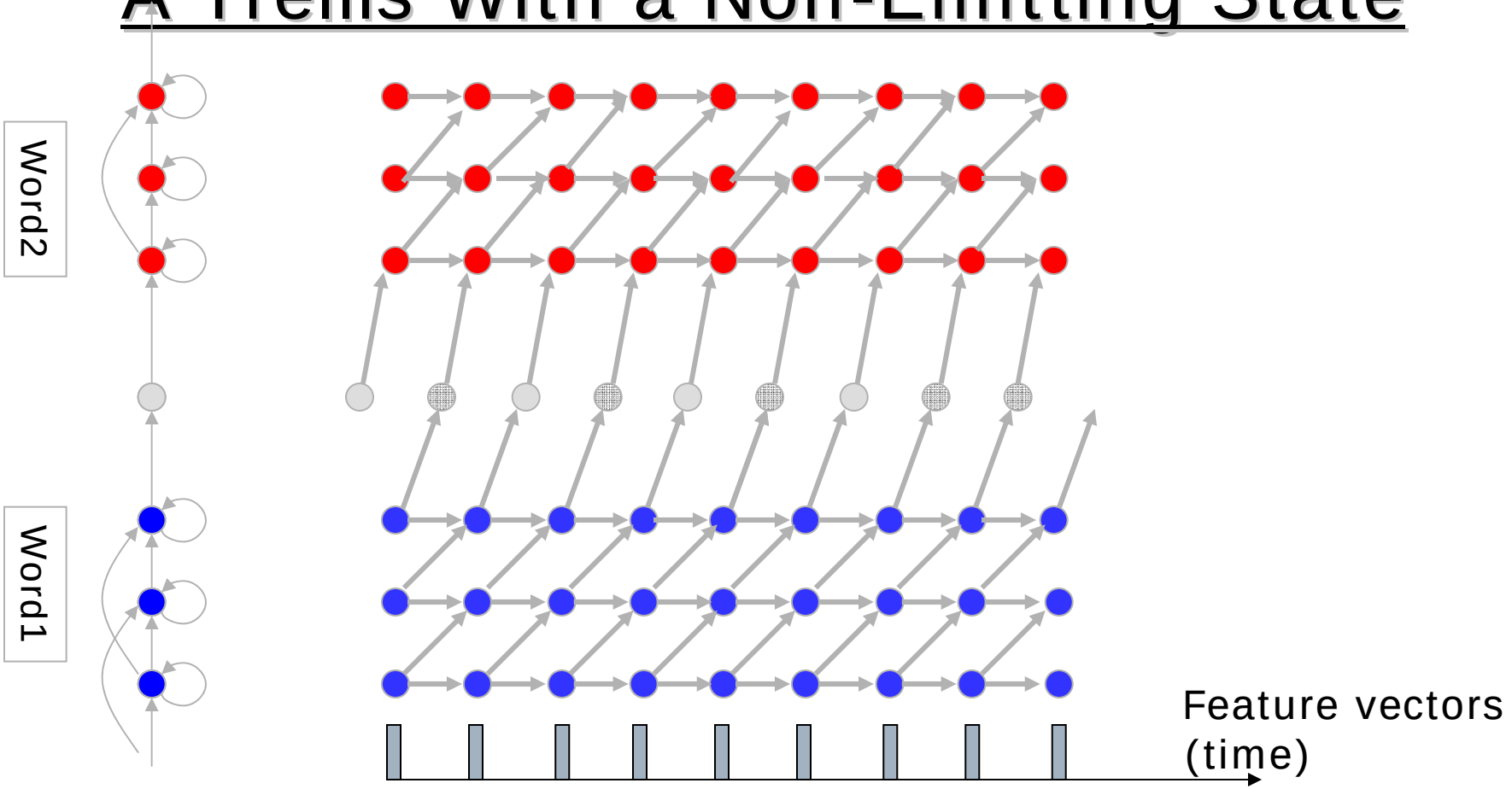
Retaining the Non-emitting State





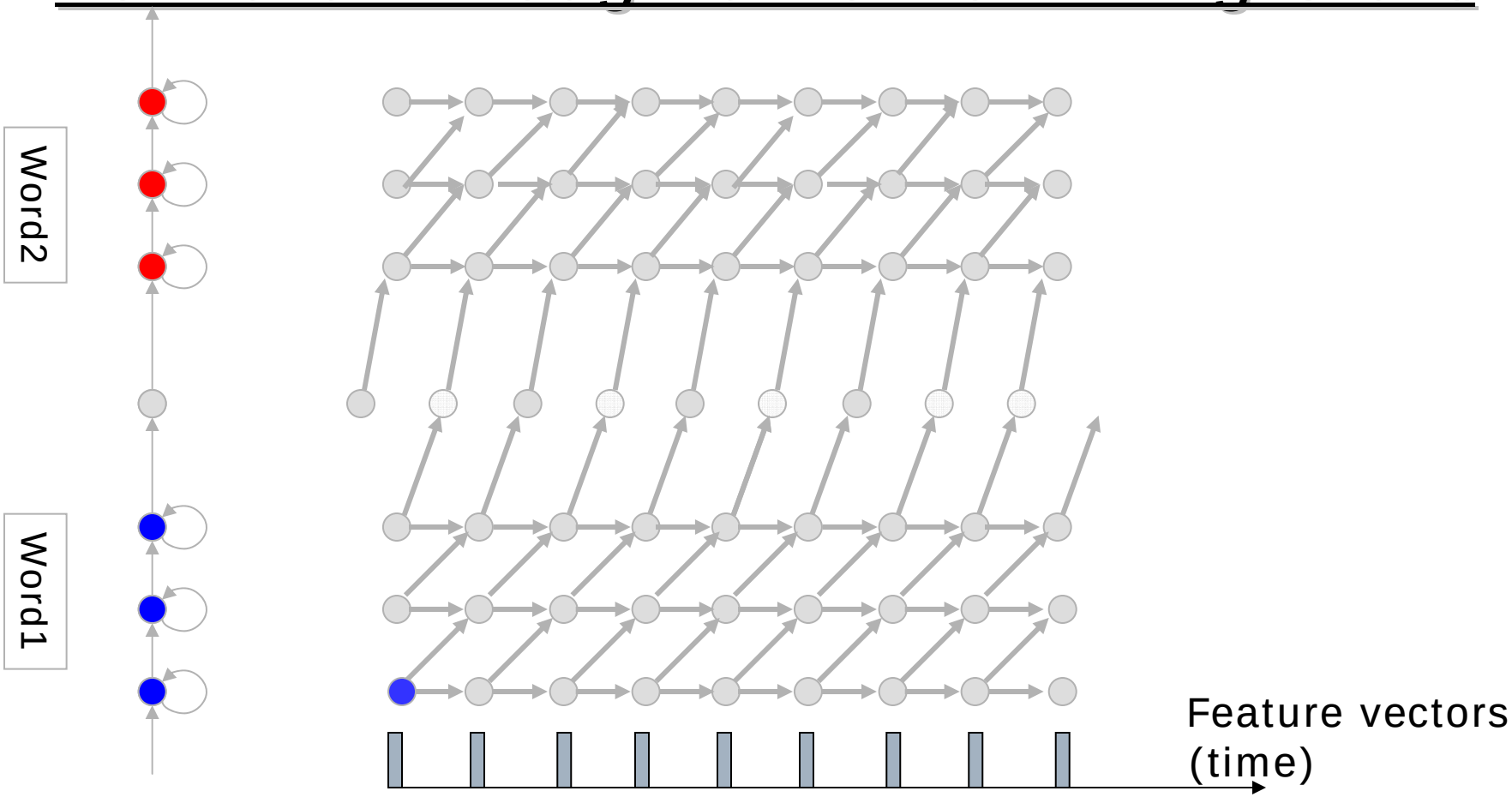
HMM for the word sequence “word2 word1”

A Trellis With a Non-Emitting State



- Since non-emitting states are not associated with observations, they have no "time"
  - In the trellis this is indicated by showing them *between* time marks
  - Non-emitting states have no horizontal edges – they are always exited instantly

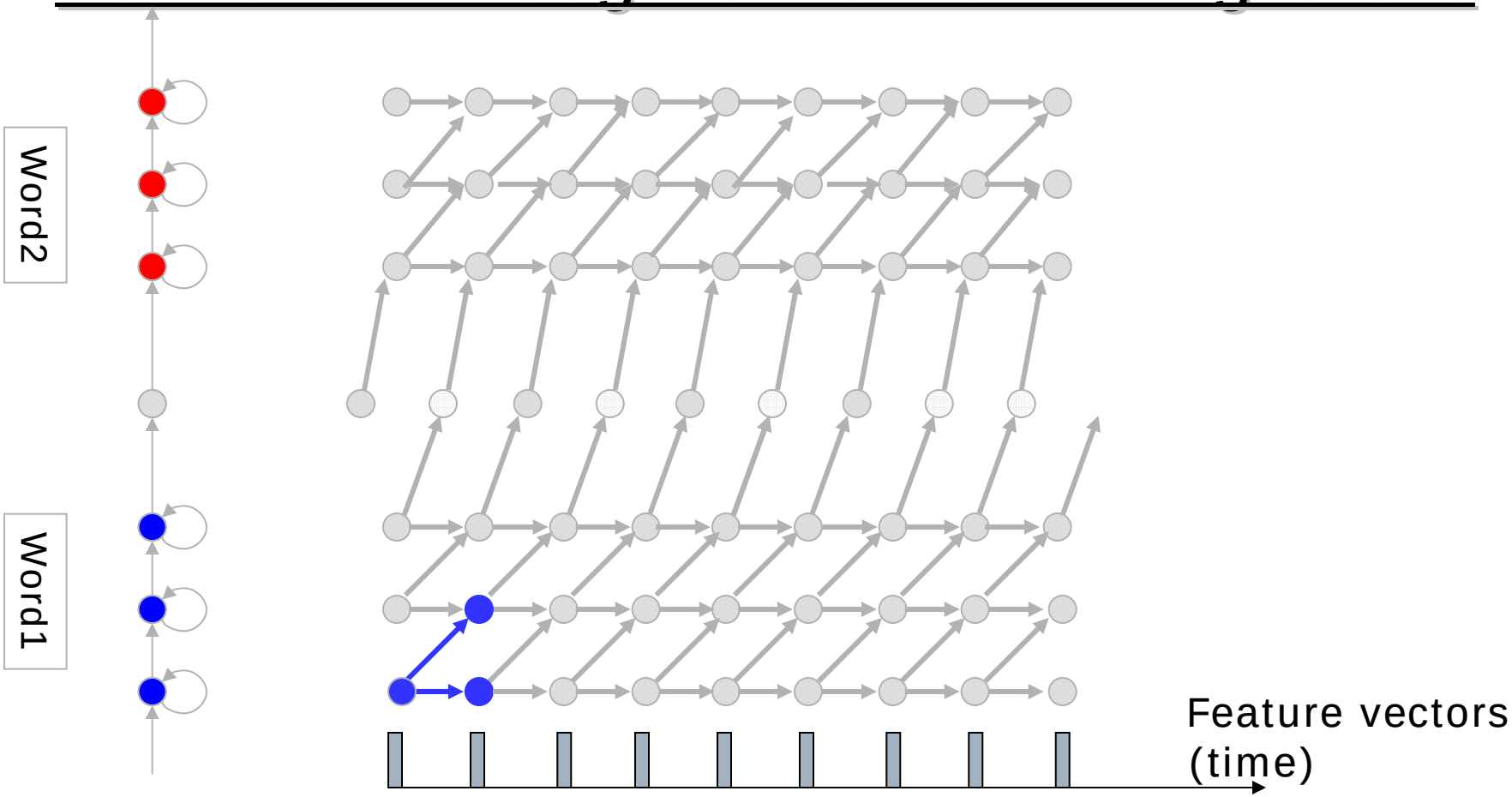
Forward Through a non-emitting State



- At the first instant only one state has a non-zero forward probability

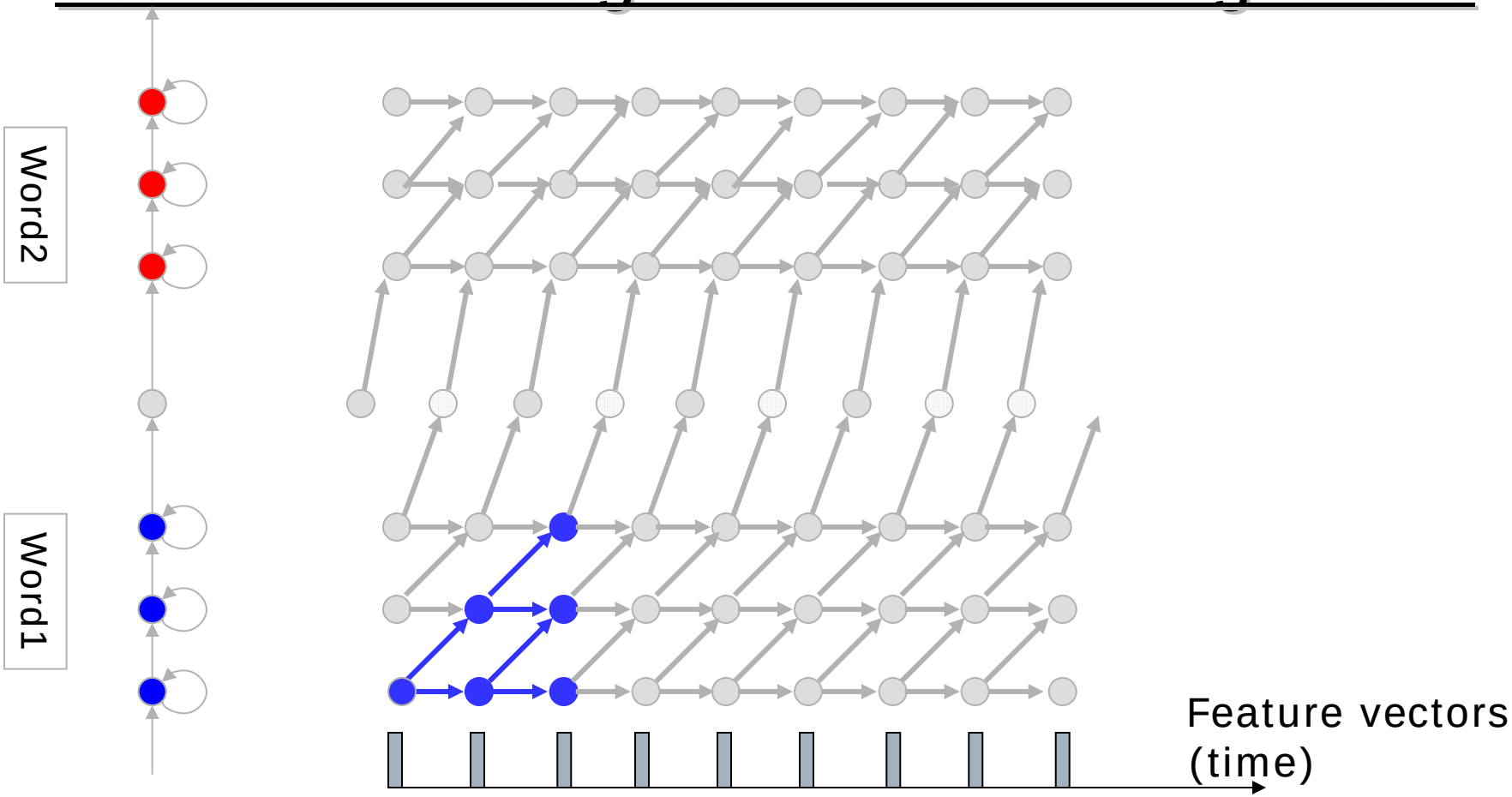
t

Forward Through a non-emitting State



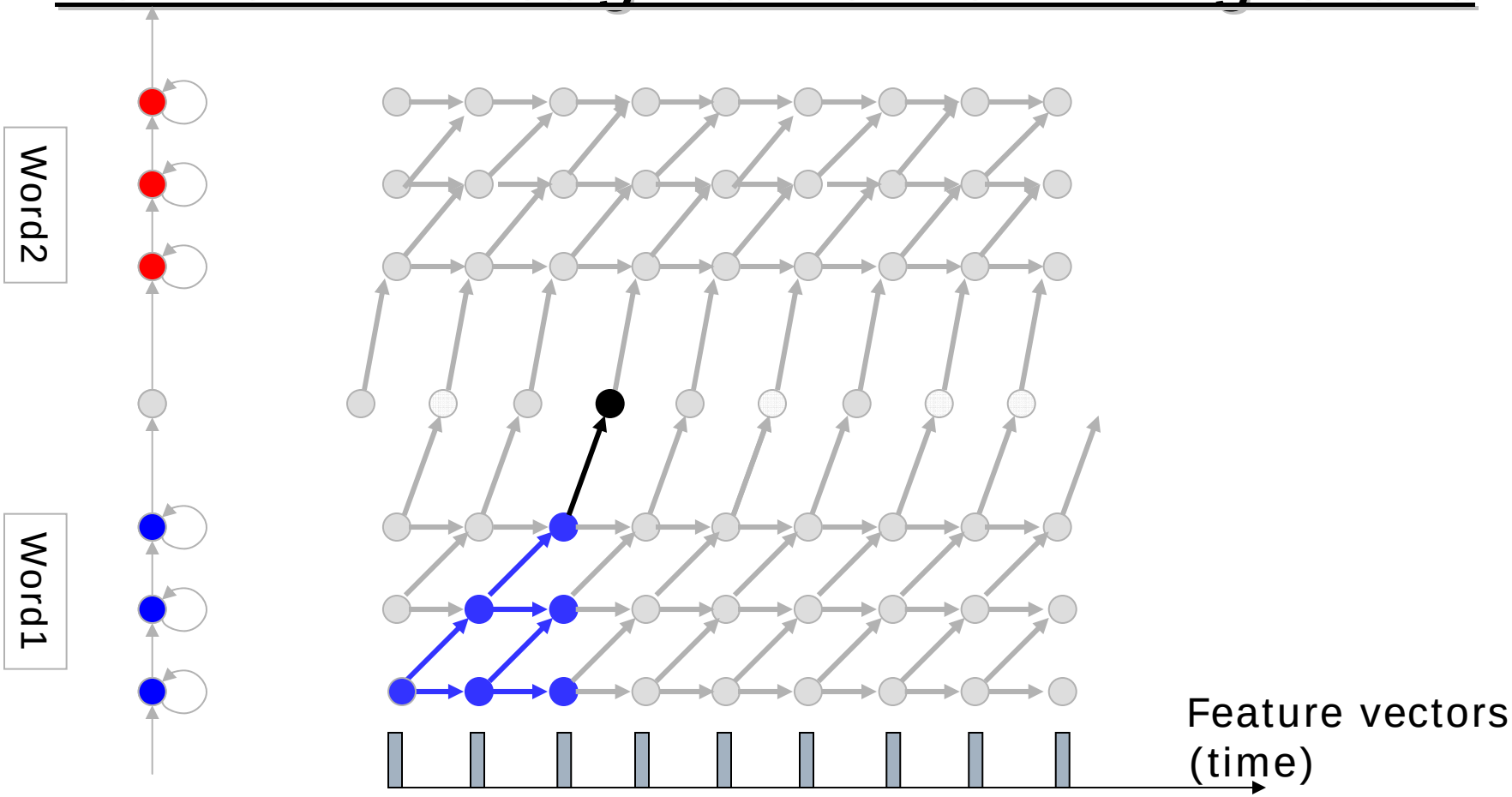
- From time 2 a number of states can have non-zero forward probabilities
  - Non-zero alphas

Forward Through a non-emitting State



- From time 2 a number of states can have non-zero forward probabilities
  - Non-zero alphas

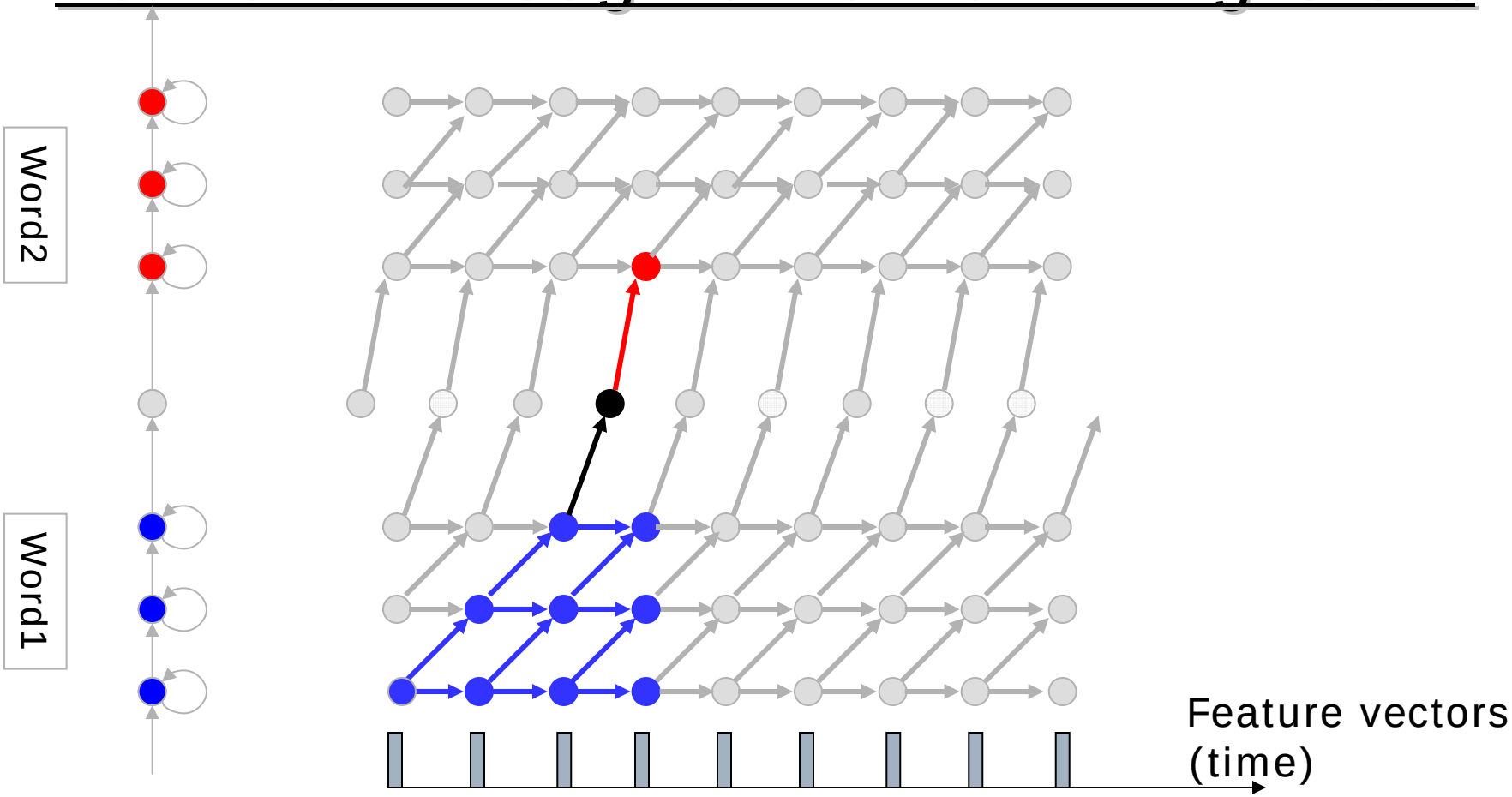
Forward Through a non-emitting State



- Between time 3 and time 4 (in this trellis) the non-emitting state gets a non-zero alpha

t

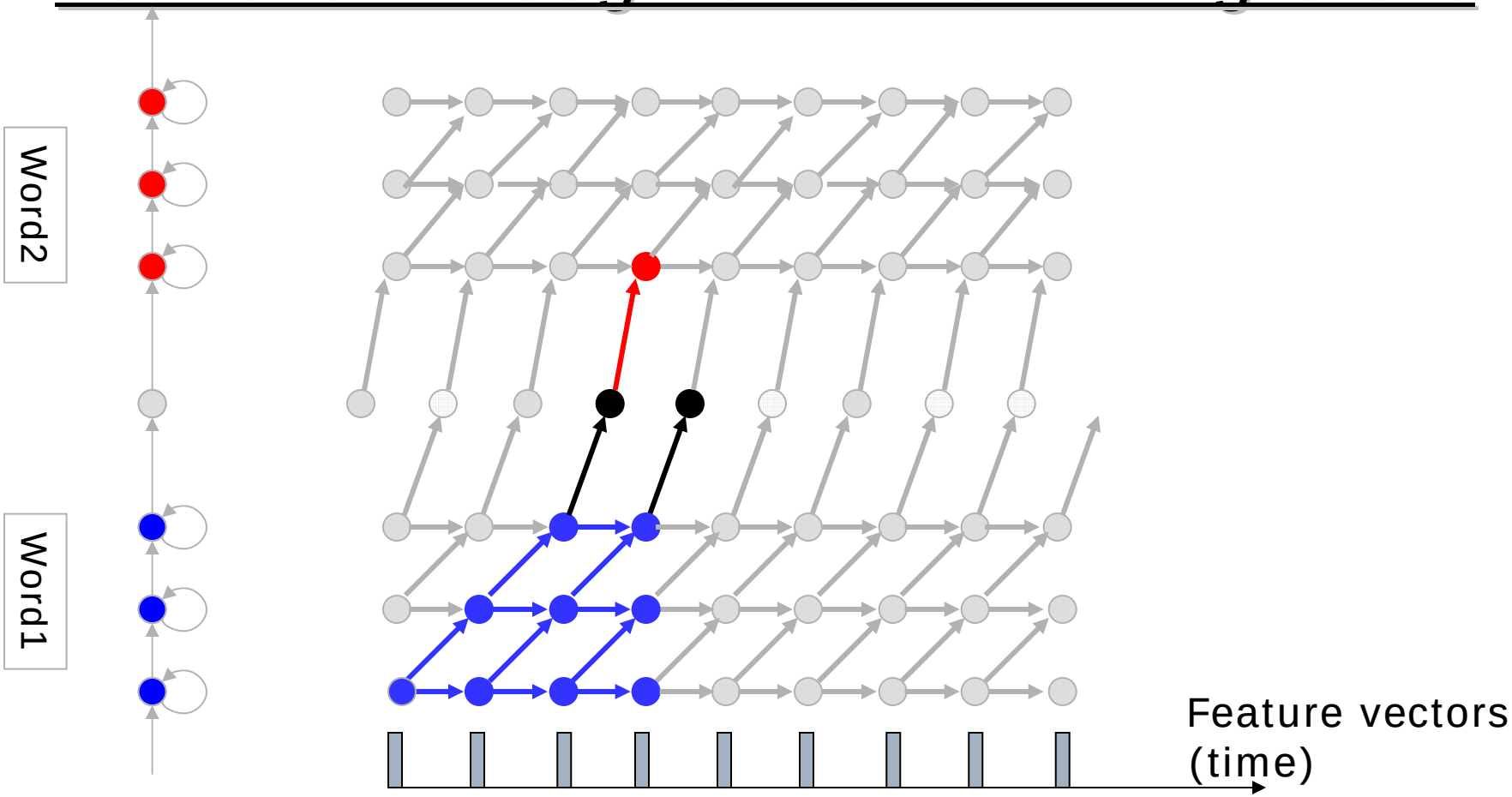
Forward Through a non-emitting State



- At time 4, the first state of word2 gets a probability contribution from the non-emitting state

t

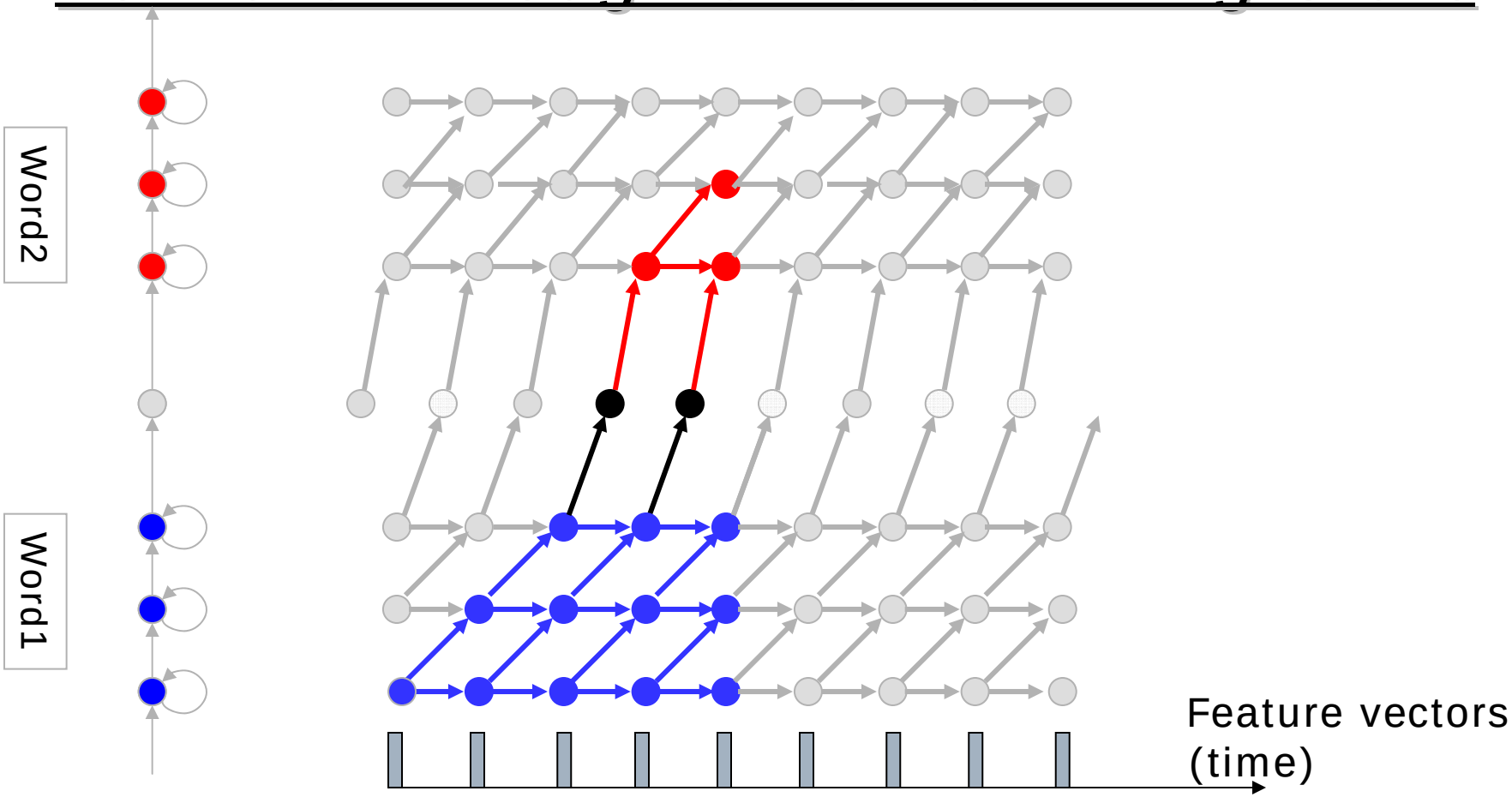
Forward Through a non-emitting State



- Between time4 and time5 the non-emitting state may be visited

t

Forward Through a non-emitting State



- At time 5 (and thereafter) the first state of word 2 gets contributions both from an emitting state (itself at the previous instant) and the non-emitting state

Forward Probability computation with non-emitting states

- The forward probability at any time has contributions from both emitting states and non-emitting states
  - This is true for both emitting states and non-emitting states.
- This results in the following rules for forward probability computation
  - Forward probability at emitting states

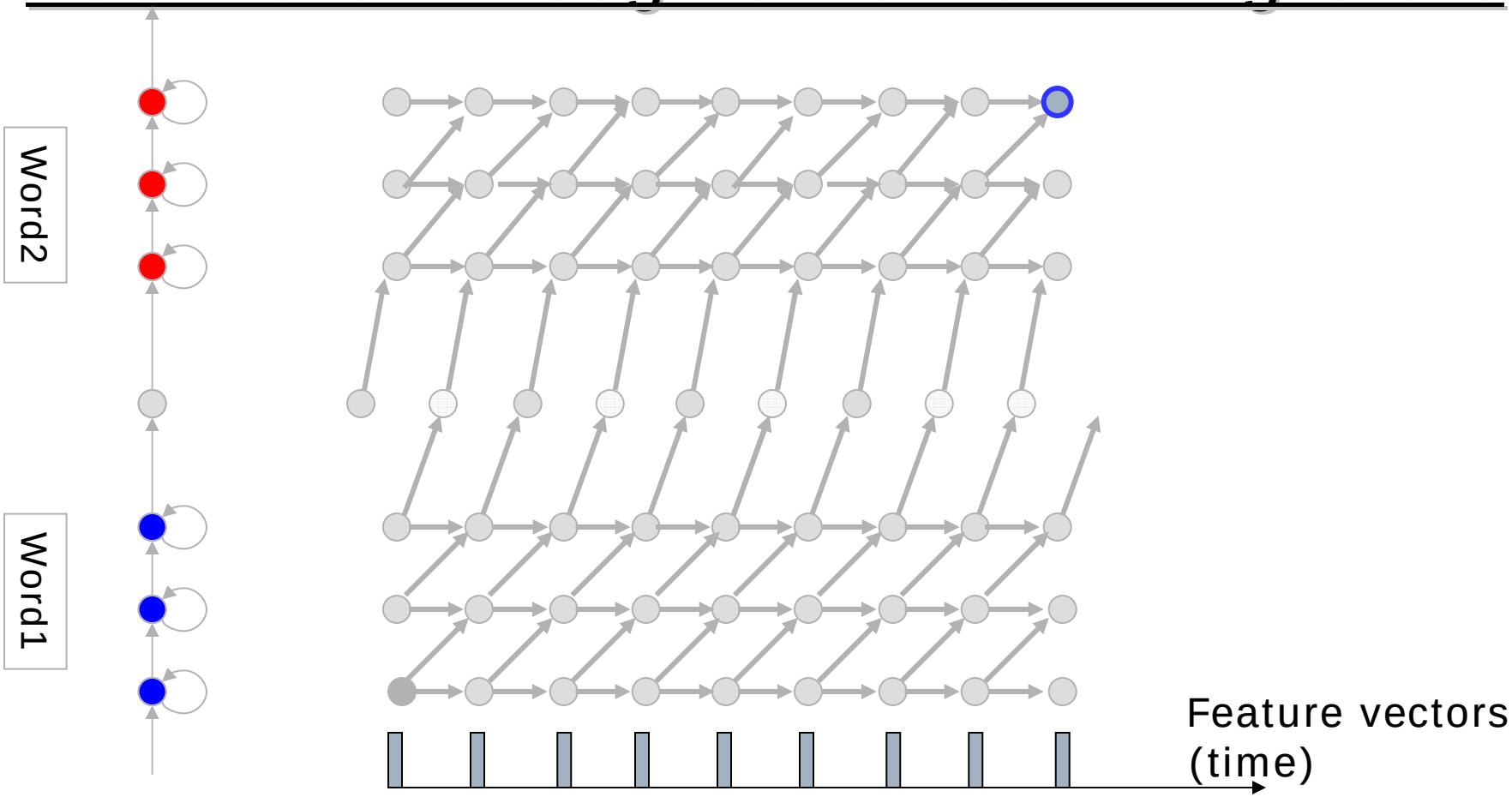
$$\alpha_u(s,t) = P(x_{u,t} \mid s)\left(\sum_{s' \in \{emitting\}} \alpha_u(s',t-1)P(s \mid s') + \sum_{s' \in \{nonemitting\}} \alpha_u(s',t)P(s \mid s') \right)$$

    - Note – although non-emitting states have no time-instant associated with them, for computation purposes they are associated with the current time

  - Forward probability at non-emitting states

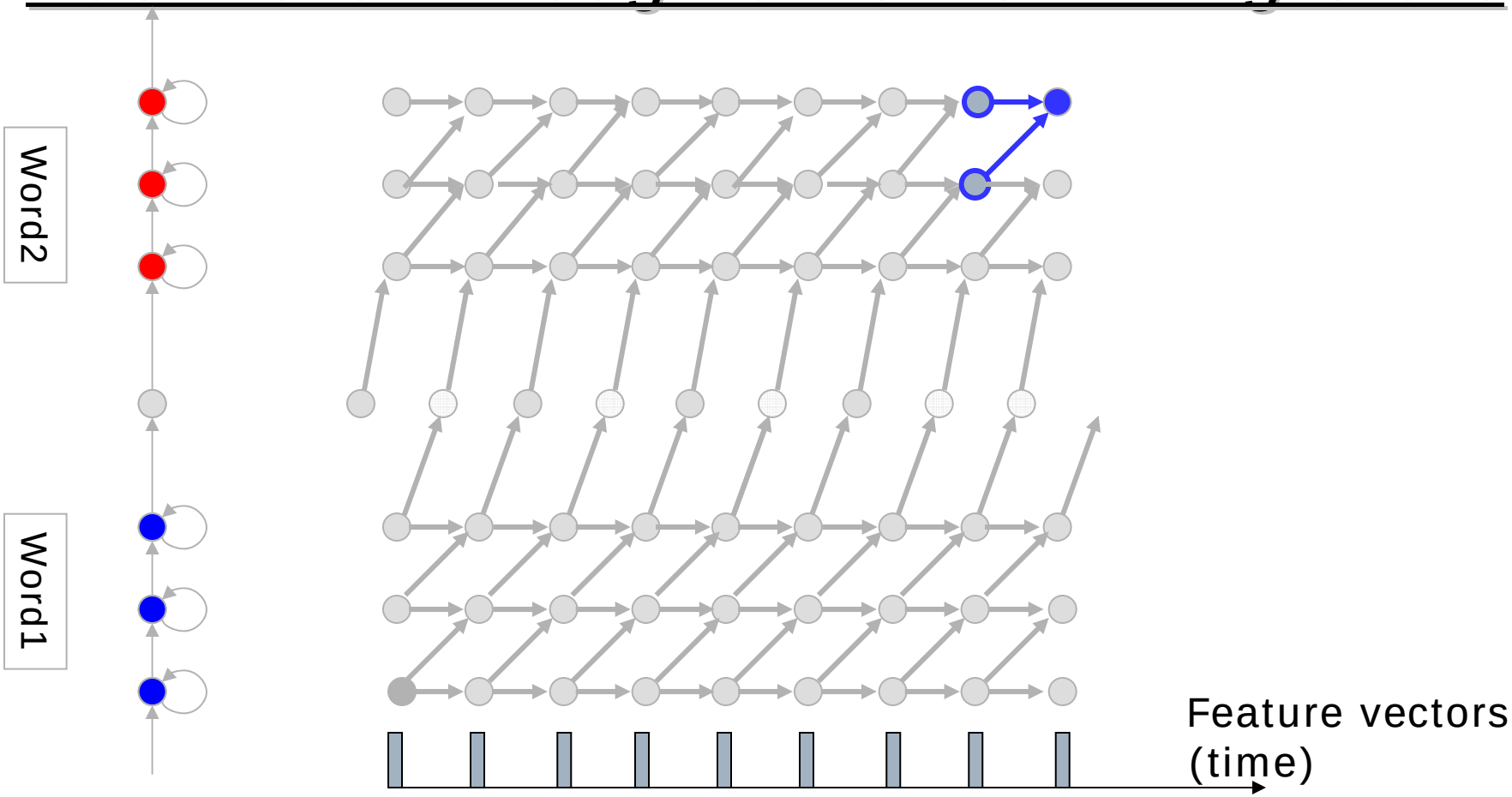
$$\alpha_u(s,t) = \sum_{s' \in \{emitting\}} \alpha_u(s',t-1)P(s \mid s') + \sum_{s' \in \{nonemitting\}} \alpha_u(s',t)P(s \mid s')$$

Backward Through a non-emitting State



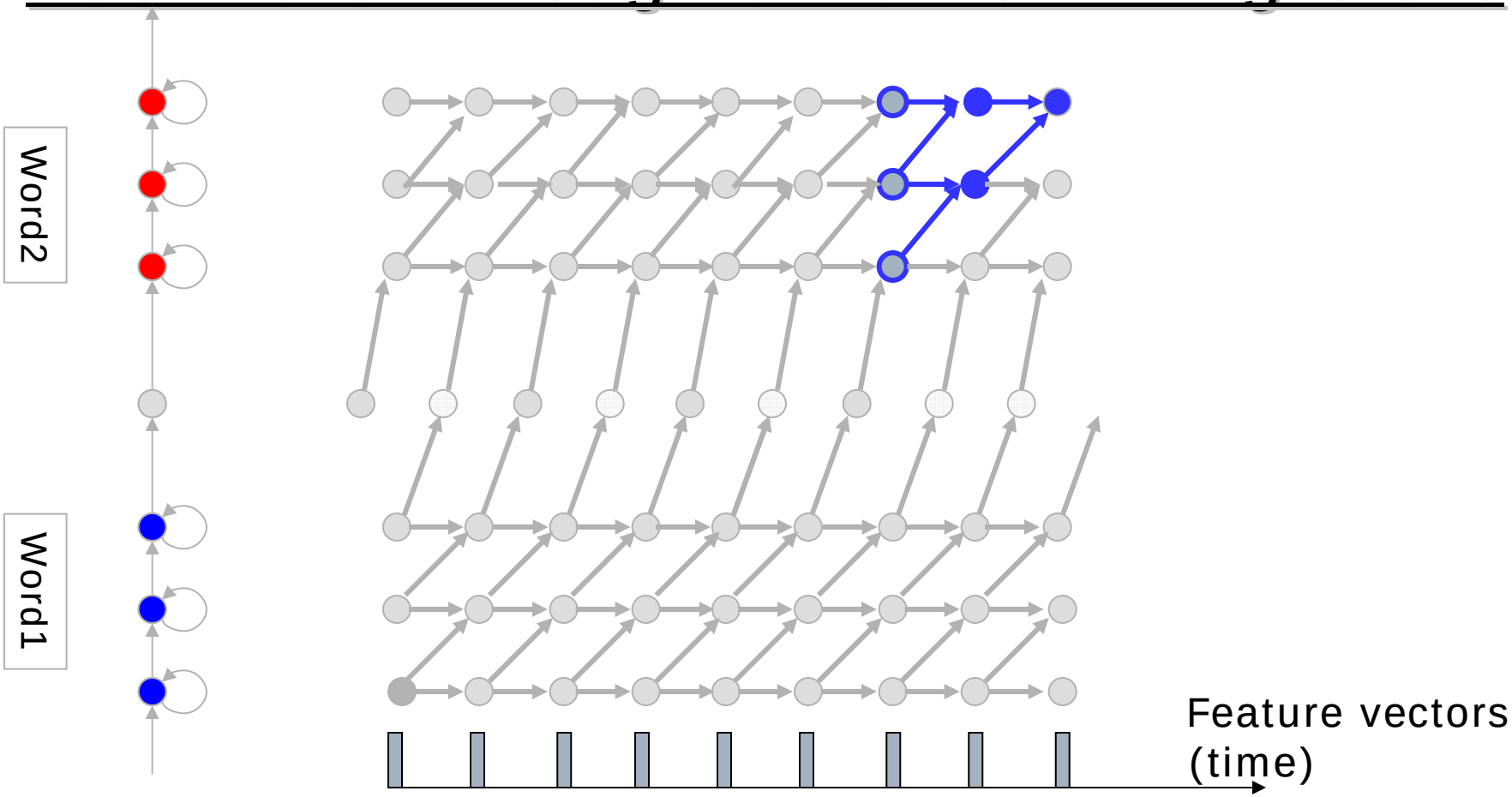
- The Backward probability has a similar property
  - States may have contributions from both emitting and non-emitting states
  - Note that current observation probability is not part of beta
    - Illustrated by grey fill in circles representing nodes

Backward Through a non-emitting State



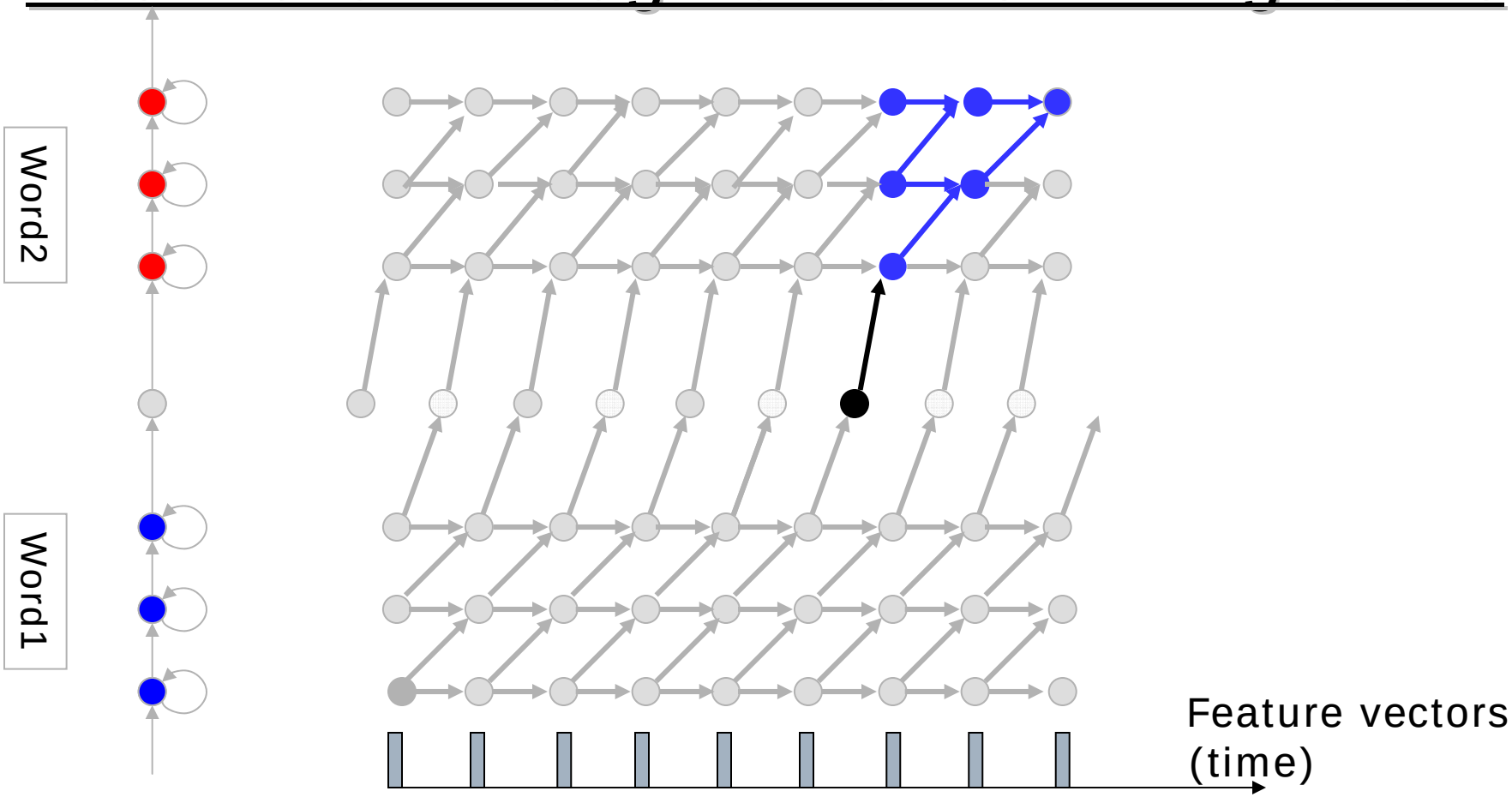
- The Backward probability has a similar property
  - States may have contributions from both emitting and non-emitting states
  - Note that current observation probability is not part of beta
    - Illustrated by grey fill in circles representing nodes

Backward Through a non-emitting State



- The Backward probability has a similar property
  - States may have contributions from both emitting and non-emitting states
  - Note that current observation probability is not part of beta
    - Illustrated by grey fill in circles representing nodes

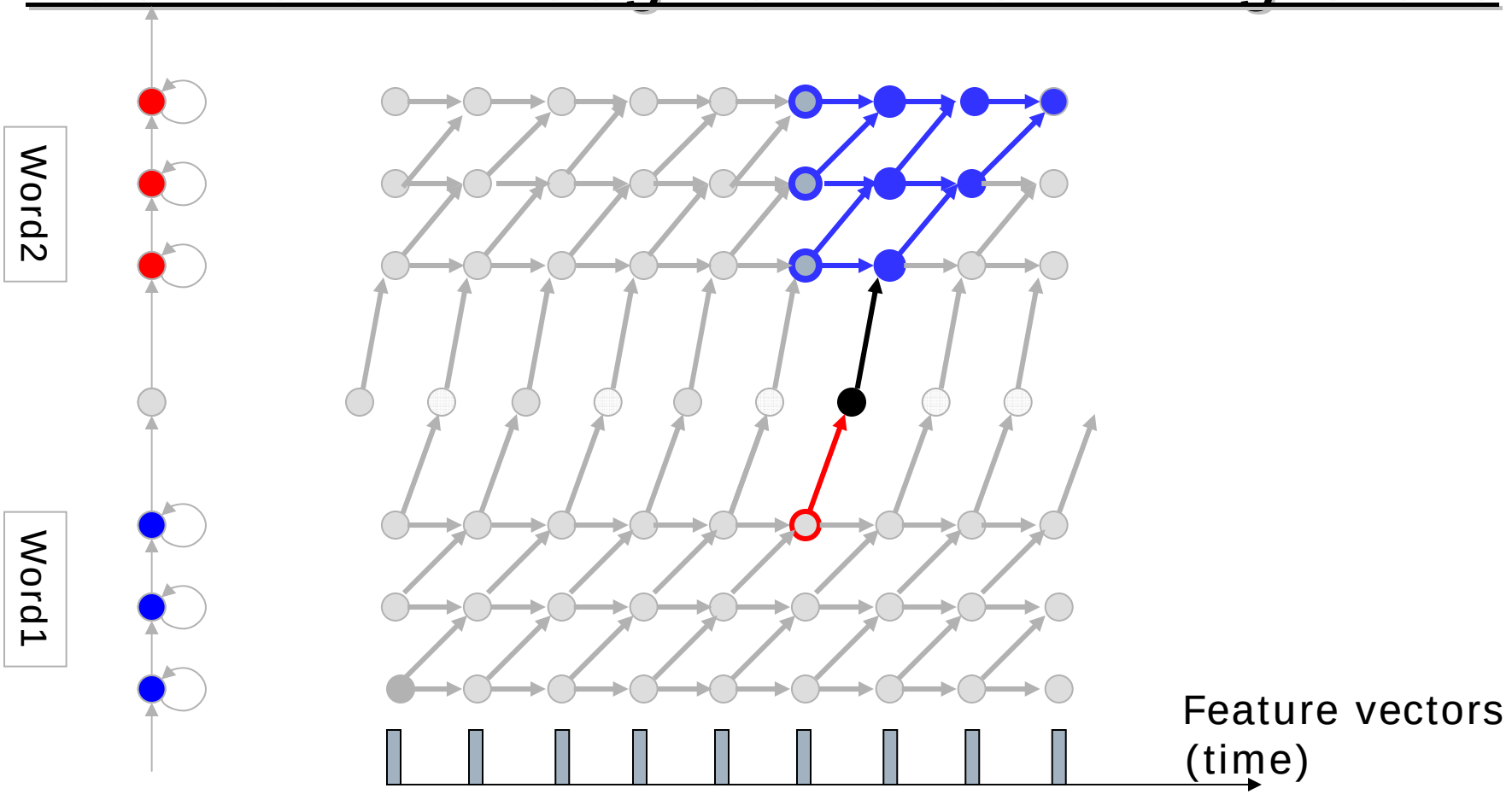
Backward Through a non-emitting State



- To activate the non-emitting state, observation probabilities of downstream observations must be factored in

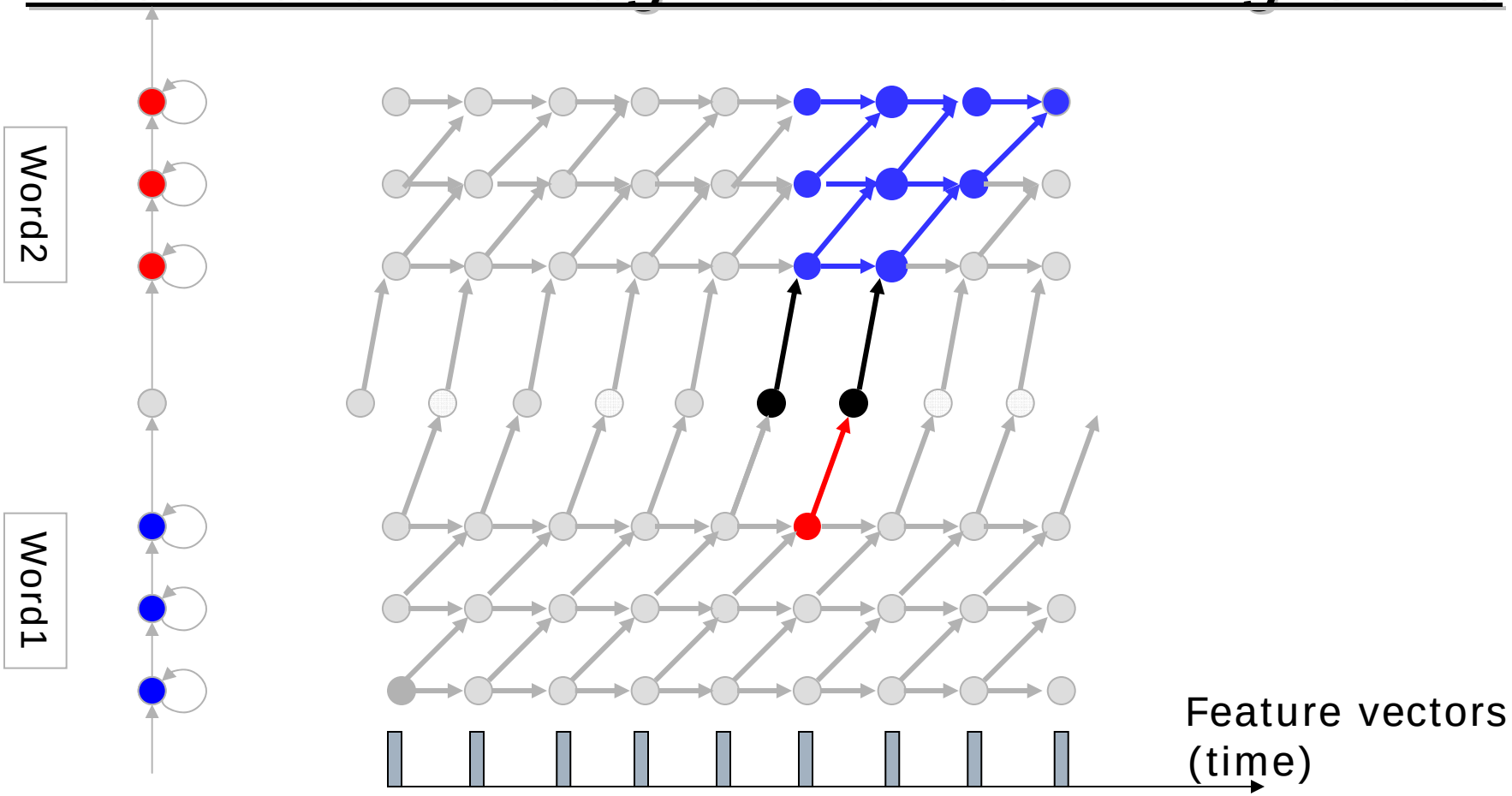
t

Backward Through a non-emitting State



- The backward probability computation proceeds past the non-emitting state into word 1.
- Observation probabilities are factored into (end-2) before the betas at (end-3) are computed

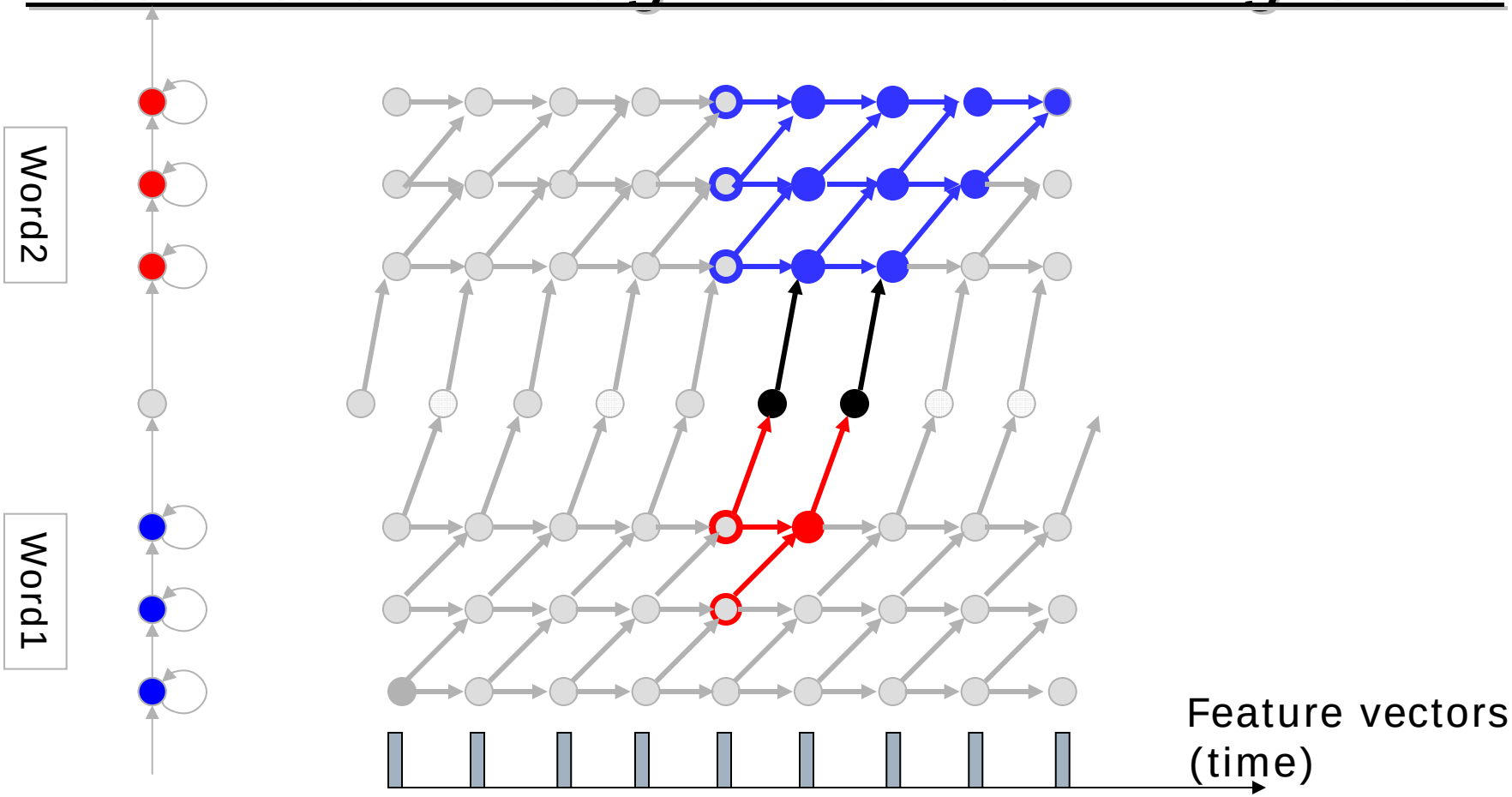
Backward Through a non-emitting State



- Observation probabilities at (end-3) are still factored into the beta for the non-emitting state between (end-3) and (end-4)

t

Backward Through a non-emitting State



- Backward probabilities at (end-4) have contributions from both future emitting states and non-emitting states

t

Backward Probability computation with non-emitting states

- The backward probability at any time has contributions from both emitting states and non-emitting states
  - This is true for both emitting states and non-emitting states.

- Since the backward probability does not factor in current observation probability, the only difference in the formulae for emitting and non-emitting states is the time stamp

  - Emitting states have contributions from emitting and non-emitting states with the next timestamp

$$\beta_u(s,t) = \sum_{s' \in \{emitting\}} \beta_u(s', t+1) P(s'|s) P(x_{u,t+1}|s') + \sum_{s' \in \{nonemitting\}} \beta_u(s', t+1) P(s'|s)$$

  - Non-emitting states have contributions from other states with the same time stamp

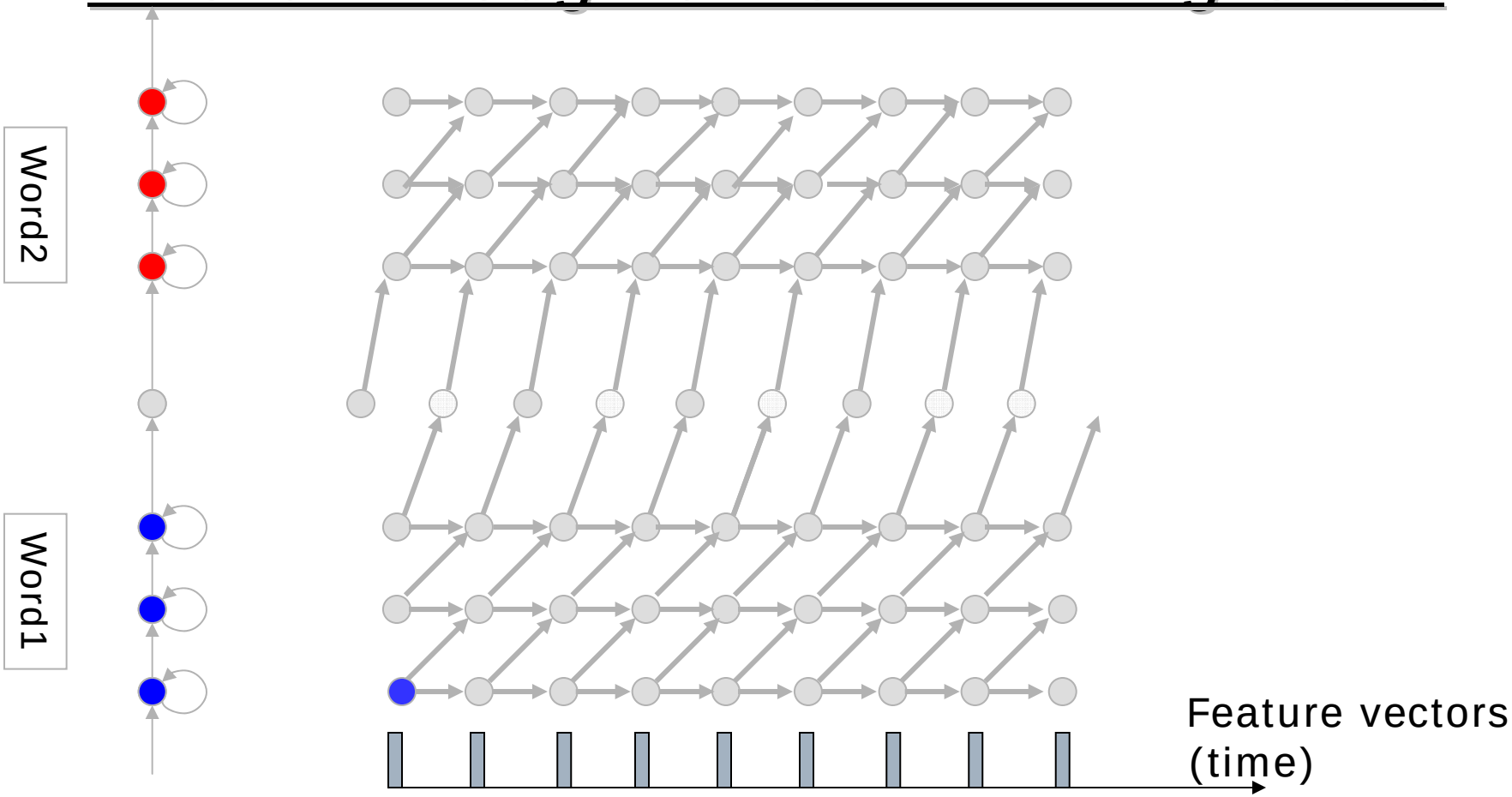
$$\beta_u(s,t) = \sum_{s' \in \{emitting\}} \beta_u(s', t) P(s'|s) P(x_{u,t}|s') + \sum_{s' \in \{nonemitting\}} \beta_u(s', t) P(s'|s)$$

Detour: Viterbi with Non-emitting States

- Non-emitting states affect Viterbi decoding
  - The process of obtaining state segmentations

- This is critical for the actual recognition algorithm for word sequences

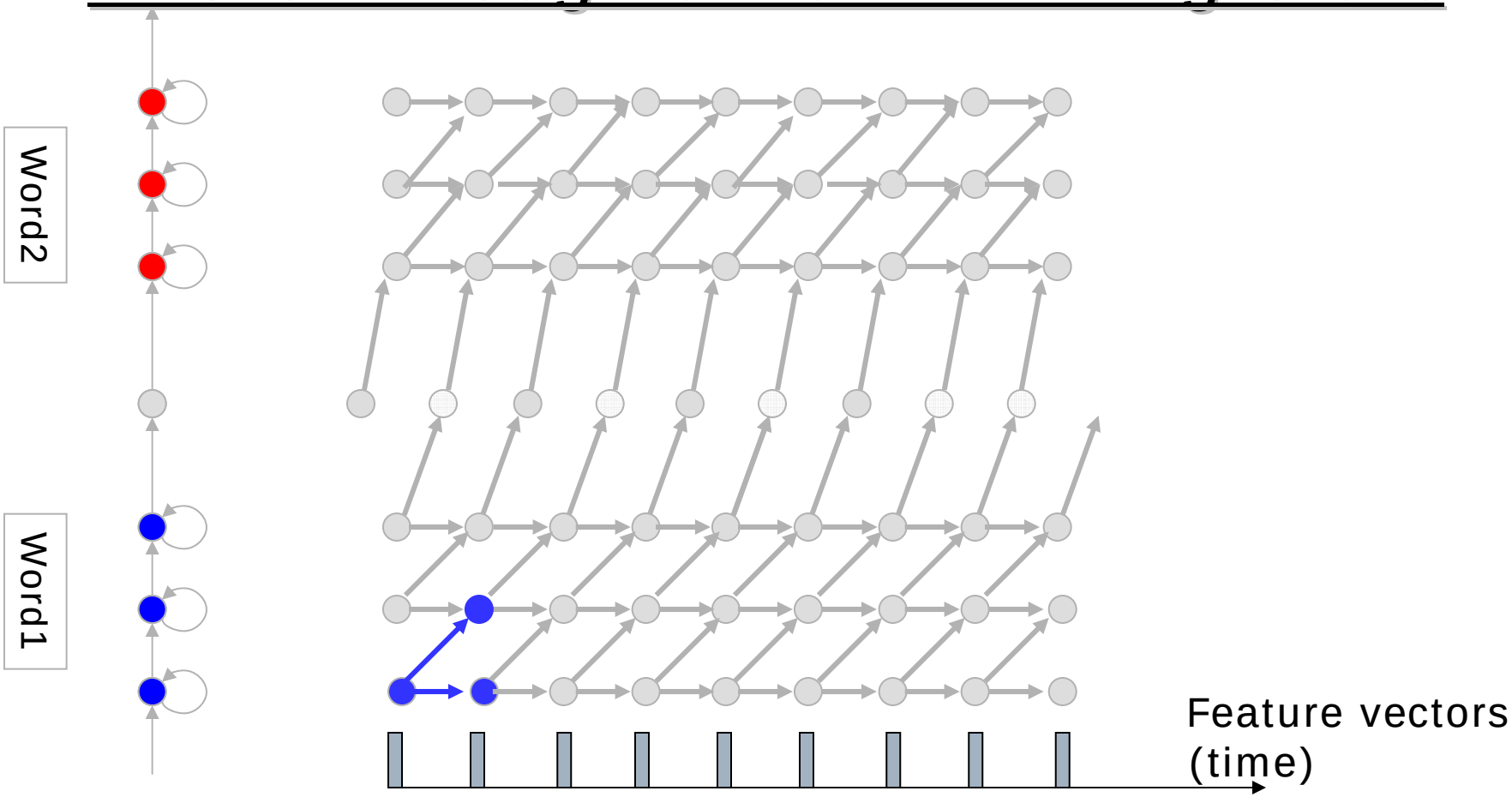
Viterbi through a Non-Emitting State



- At the first instant only the first state may be entered

t

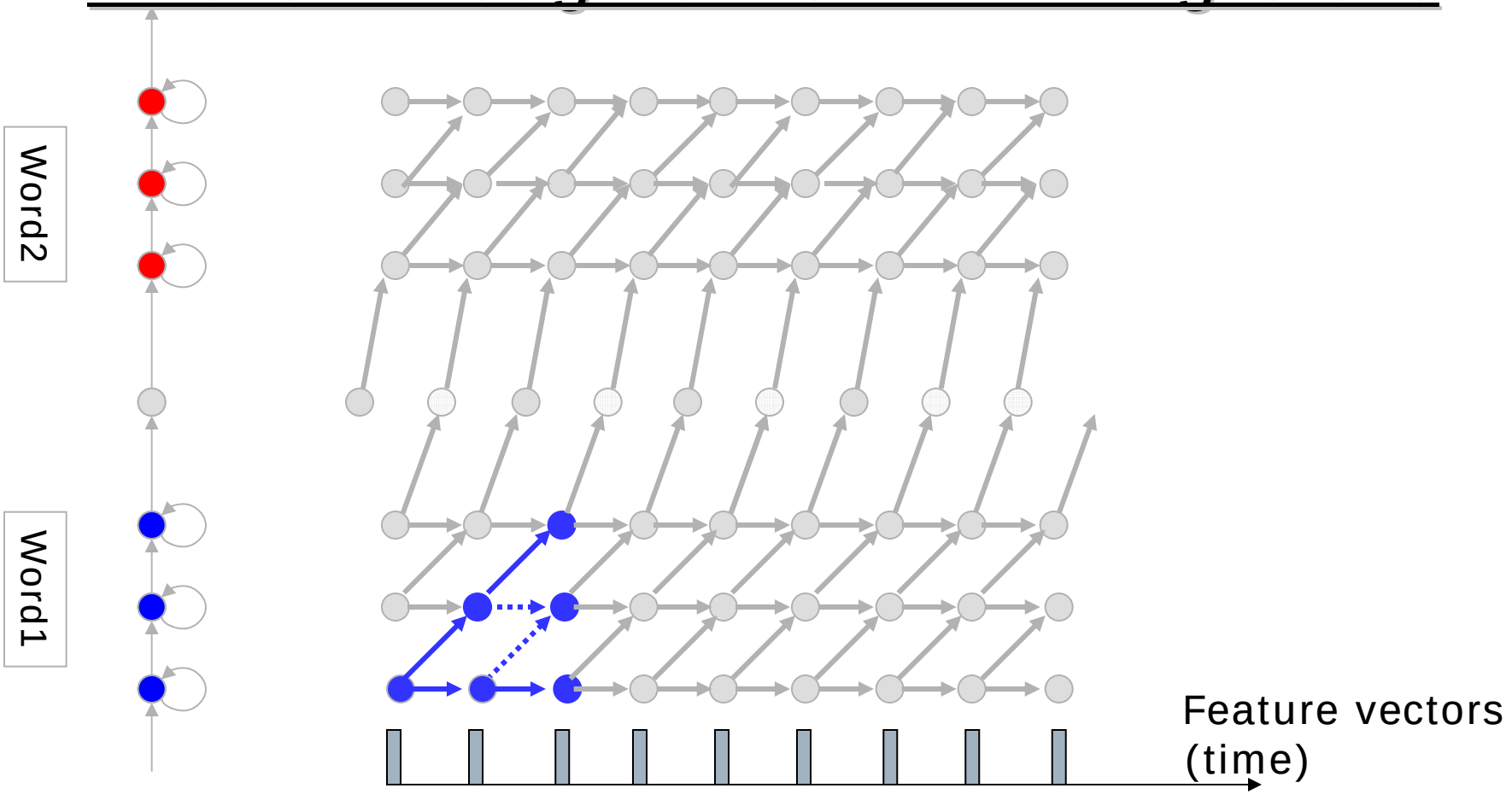
Viterbi through a Non-Emitting State



- At t=2 the first two states have only one possible entry path

t

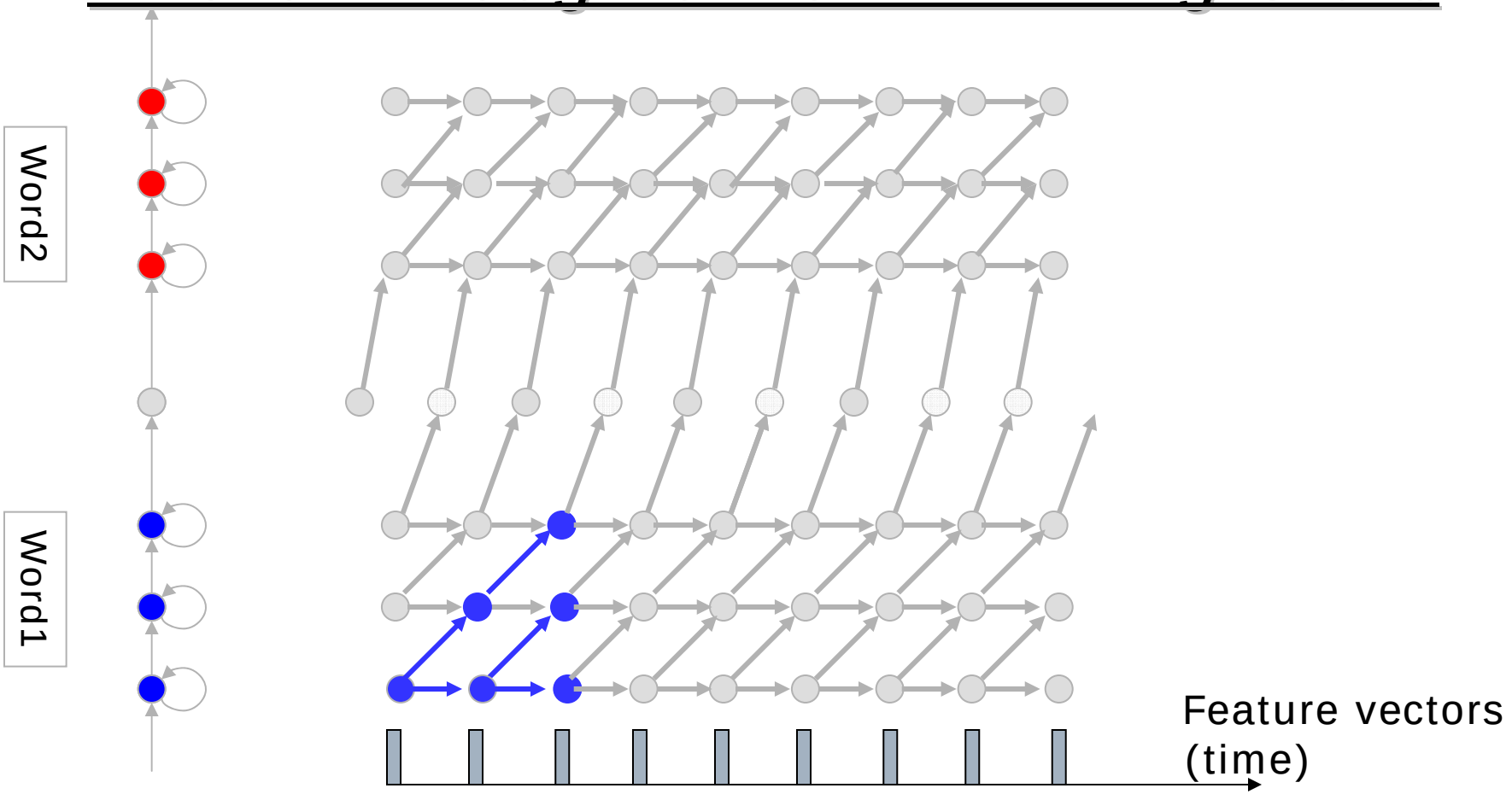
Viterbi through a Non-Emitting State



- At t=3 state 2 has two possible entries. The best one must be selected

t

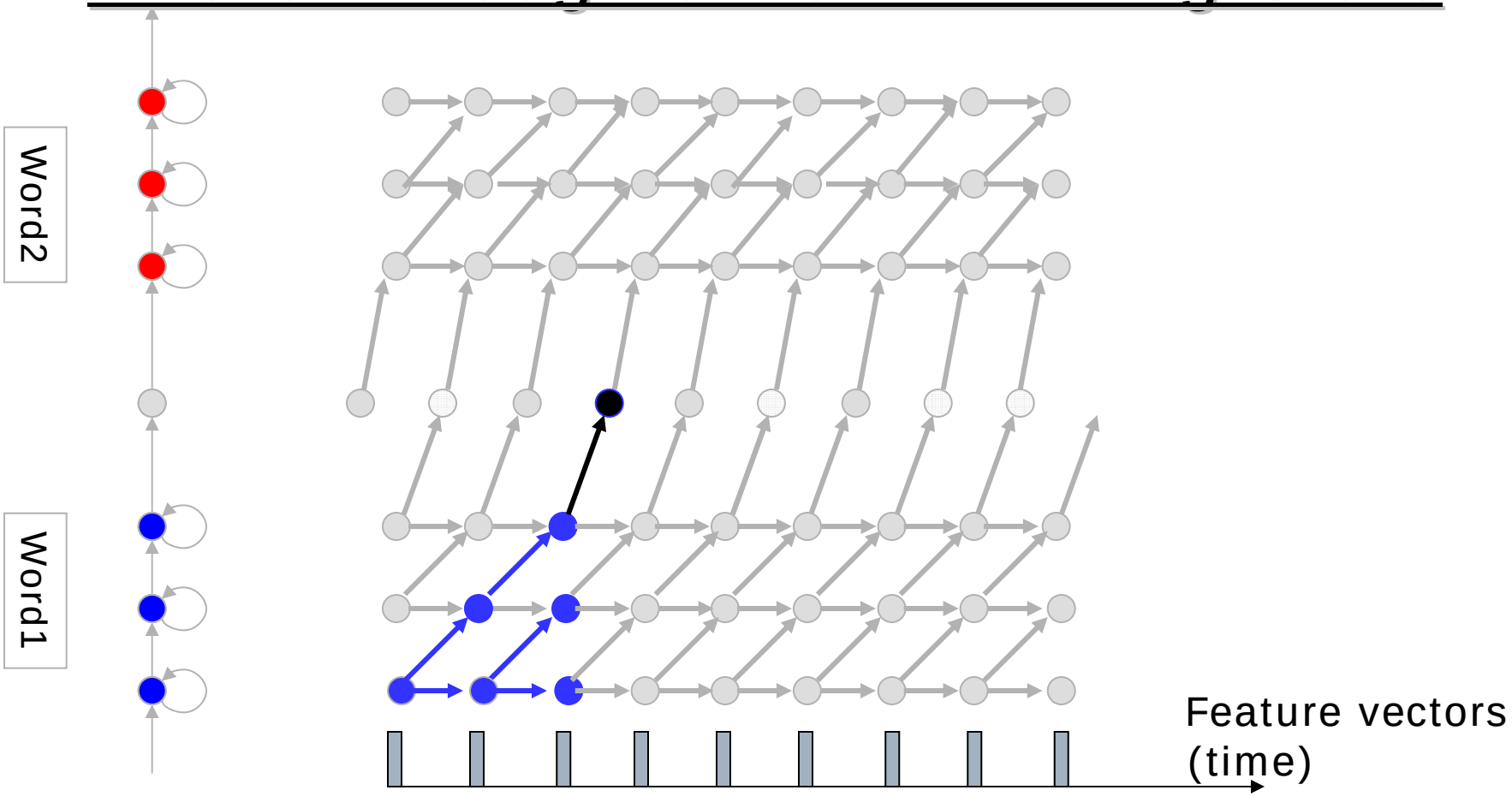
Viterbi through a Non-Emitting State



- At t=3 state 2 has two possible entries. The best one must be selected

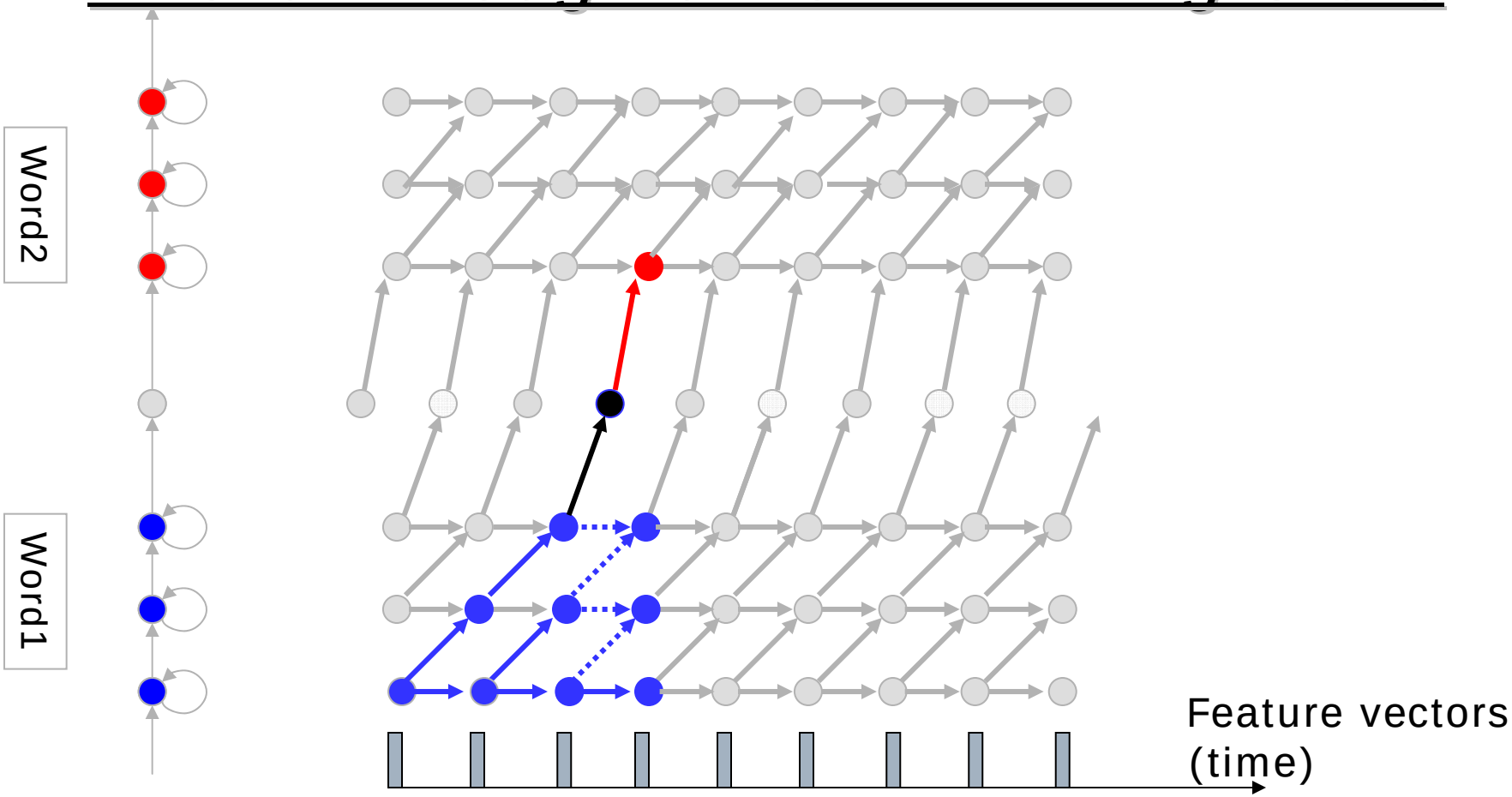
t

Viterbi through a Non-Emitting State



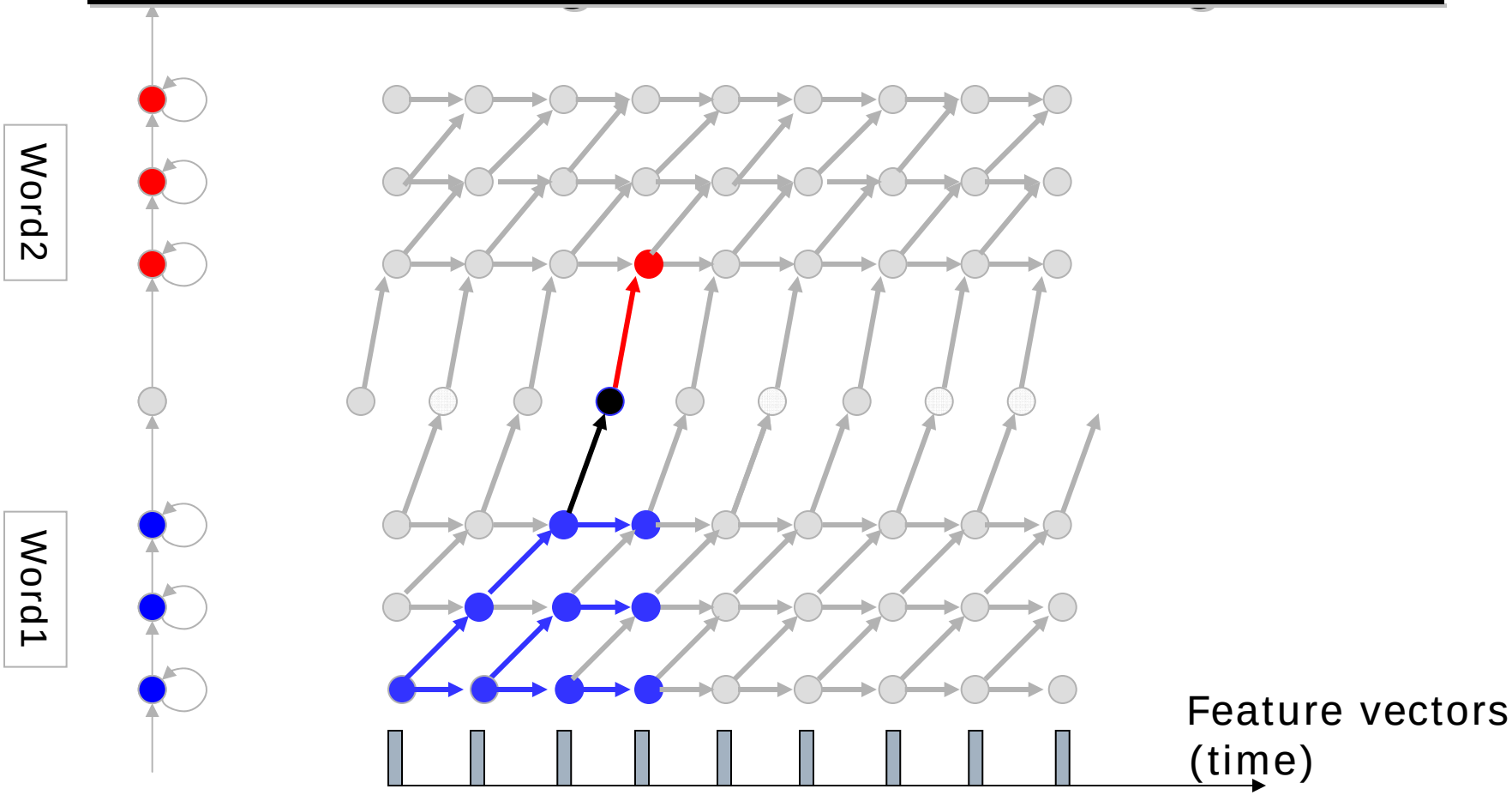
- After the third time instant we an arrive at the non-emitting state. Here there is only one way to get to the non-emitting state

Viterbi through a Non-Emitting State



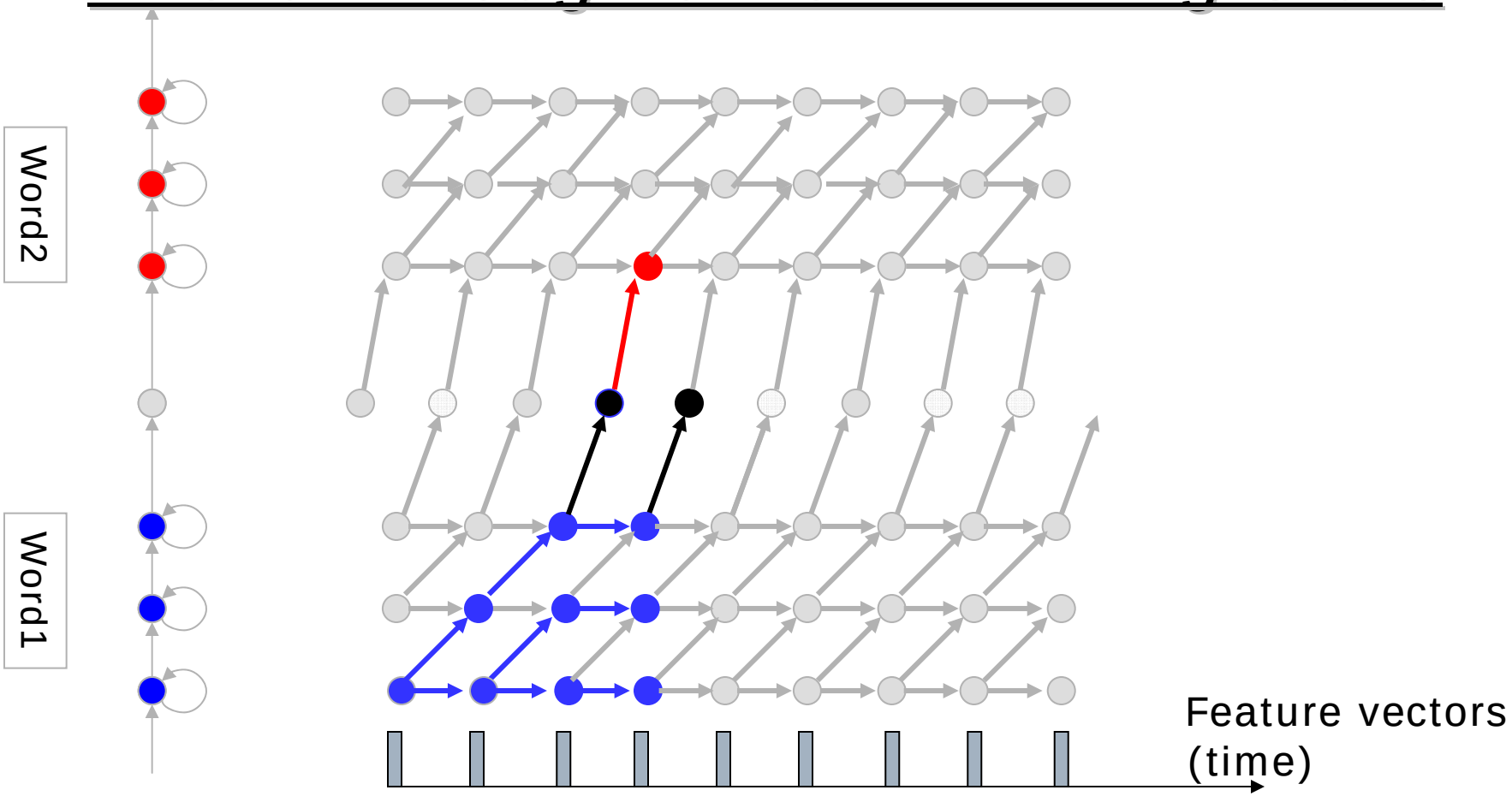
- Paths exiting the non-emitting state are now in word2
  - States in word1 are still active
  - These represent paths that have not crossed over to word2

Viterbi through a Non-Emitting State



- Paths exiting the non-emitting state are now in word2
  - States in word1 are still active
  - These represent paths that have not crossed over to word2

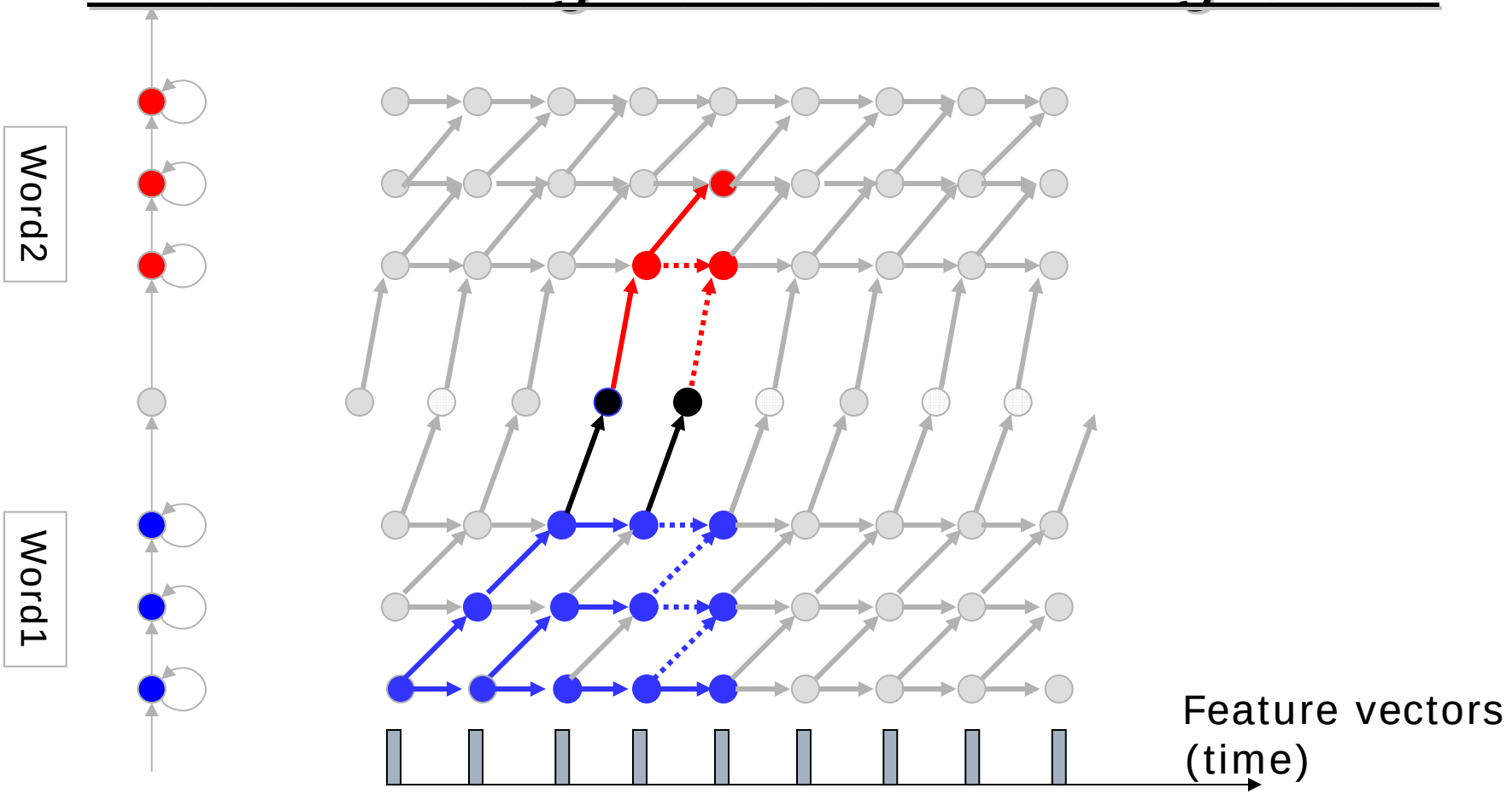
Viterbi through a Non-Emitting State



- The non-emitting state will now be arrived at after every observation instant

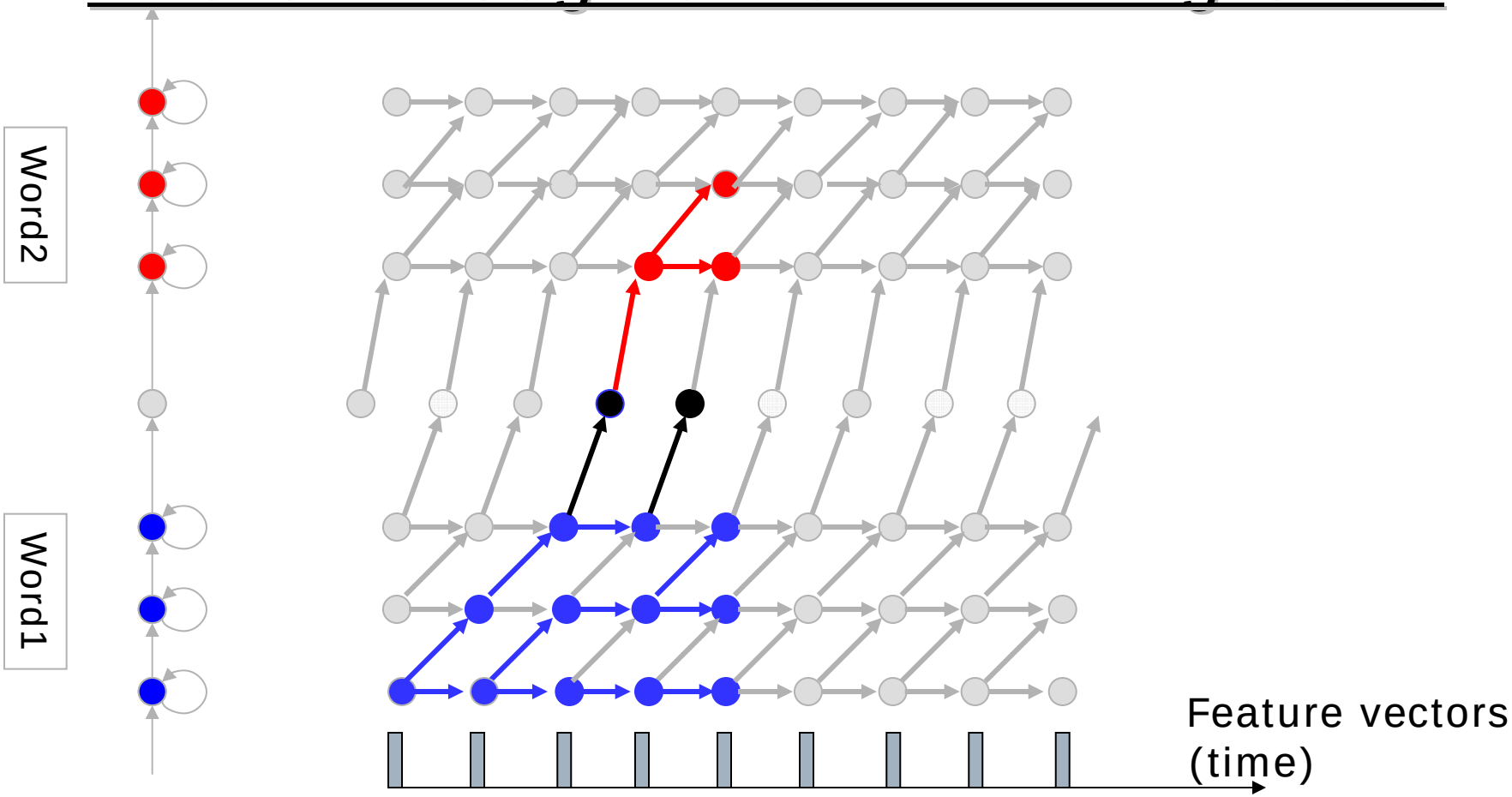
t

Viterbi through a Non-Emitting State



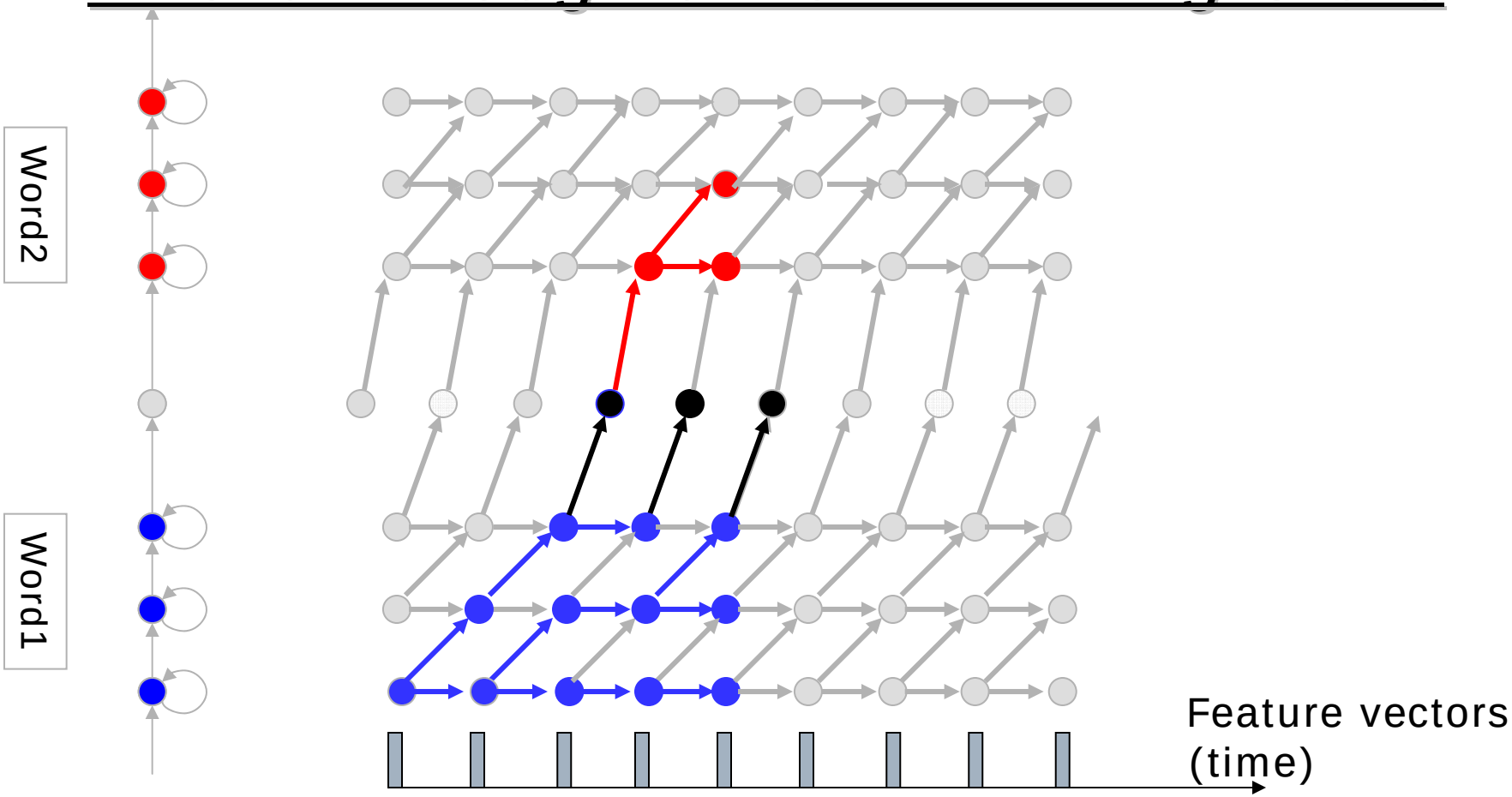
- "Enterable" states in word2 may have incoming paths either from the "cross-over" at the non-emitting state or from within the word
  - Paths from non-emitting states may compete with paths from emitting states

Viterbi through a Non-Emitting State



- Regardless of whether the competing incoming paths are from emitting or non-emitting states, the best overall path is selected

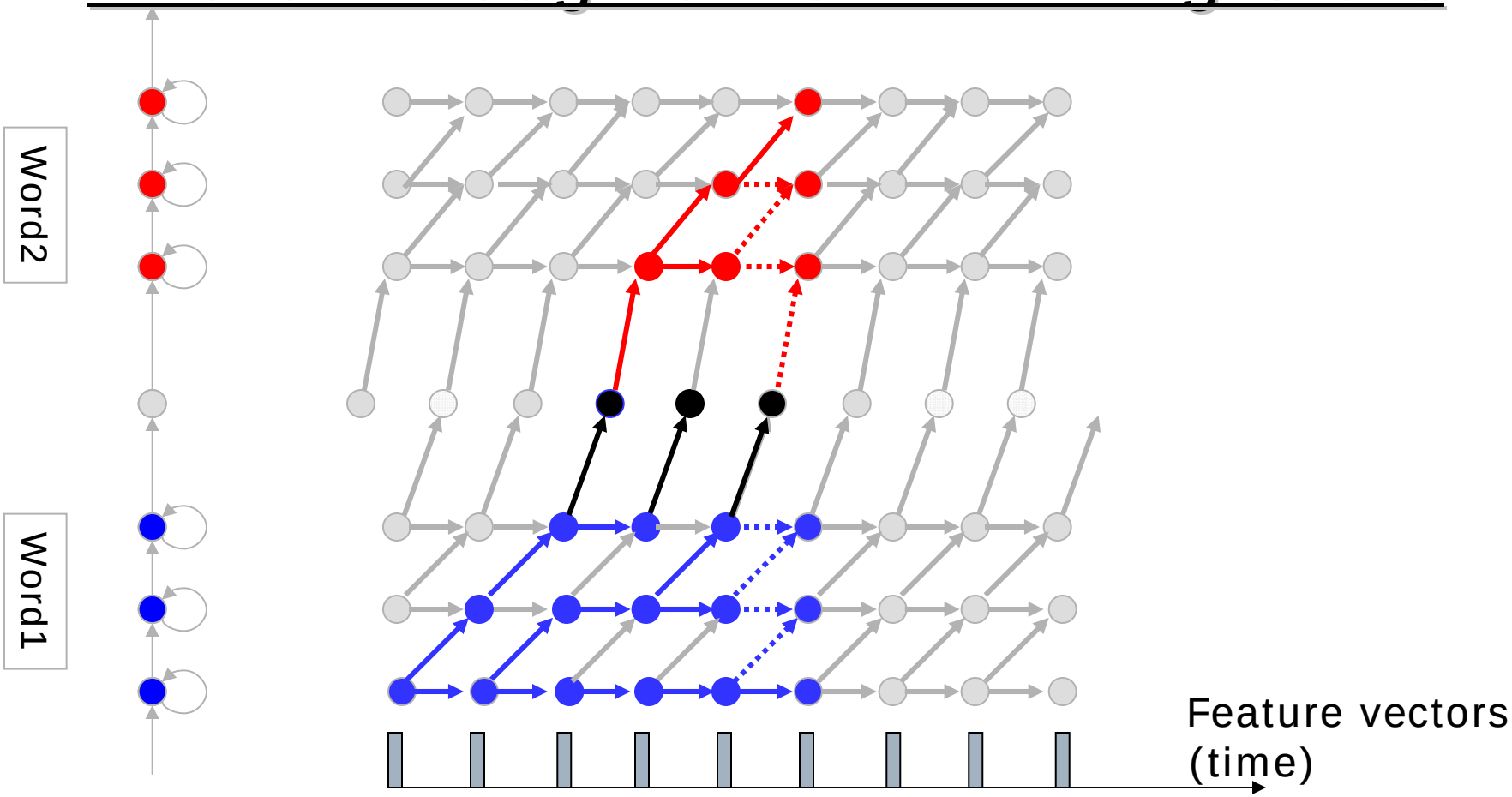
Viterbi through a Non-Emitting State



- The non-emitting state can be visited after every observation

t

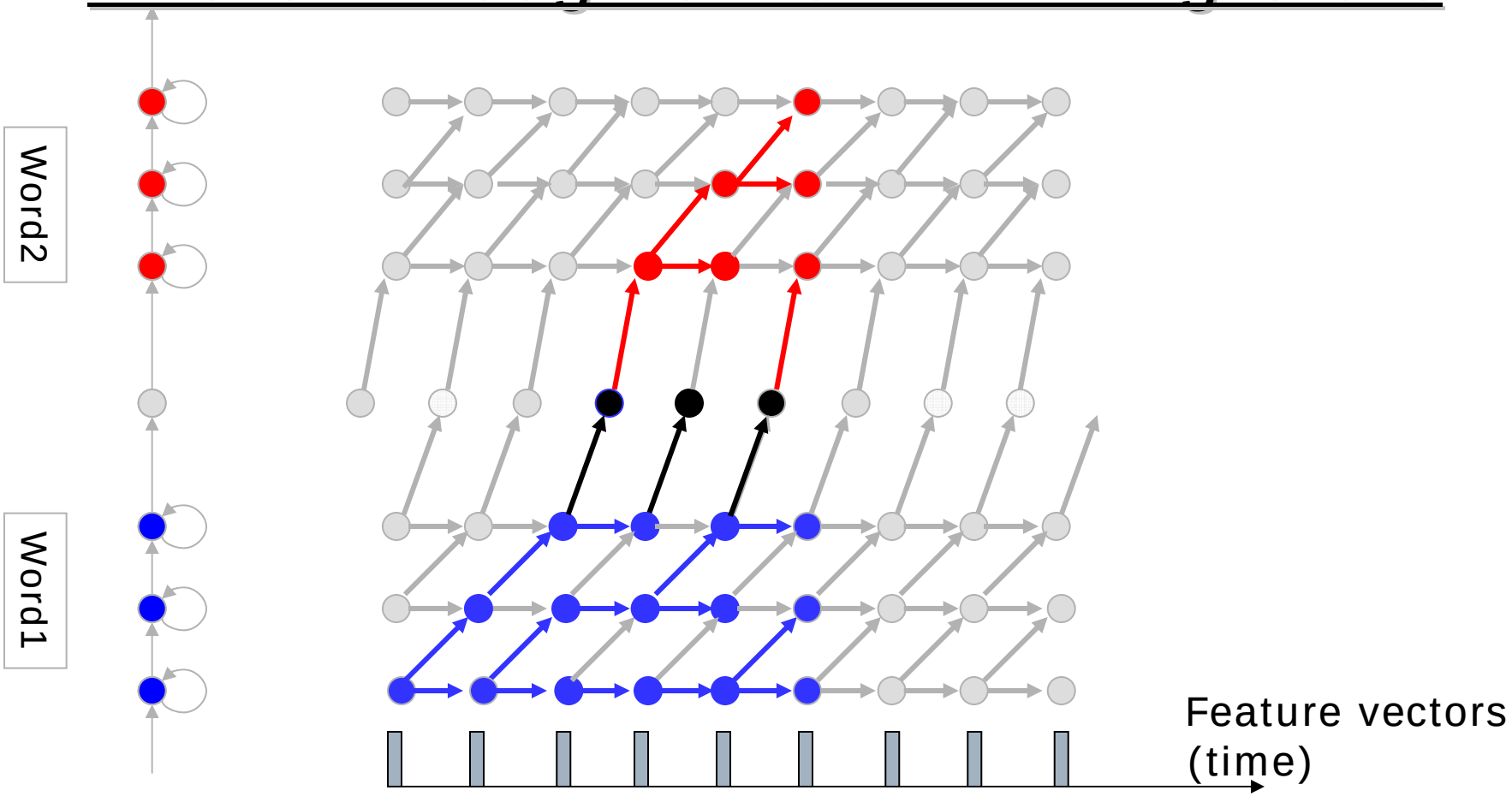
Viterbi through a Non-Emitting State



- At all times paths from non-emitting states may compete with paths from emitting states

t

Viterbi through a Non-Emitting State



- At all times paths from non-emitting states may compete with paths from emitting states
  - The best will be selected
  - This may be from either an emitting or non-emitting state

Viterbi with NULL states

- Competition between incoming paths from emitting and non-emitting states may occur at both emitting and non-emitting states
- The best path logic stays the same. The only difference is that the current observation probability is factored into emitting states
- Score for emitting state

$$P_u(s,t) = P(x_{u,t} \mid s) \max_{s'} \left(P_u(s',t-1) P(s \mid s') \mid_{s' \in \{emitting\}}, P_u(s',t) P(s \mid s') \mid_{s' \in \{nonemitting\}} \right)$$

- Score for non-emitting state

$$P_u(s,t) = \max_{s'} \left(P_u(s',t-1) P(s \mid s') \mid_{s' \in \{emitting\}}, P_u(s',t) P(s \mid s') \mid_{s' \in \{nonemitting\}} \right)$$

Learning with NULL states

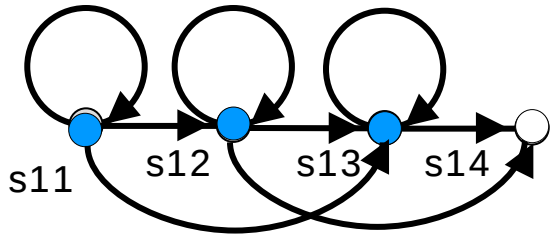
- All probability computation, state segmentation and Model learning procedures remain the same, with the previous changes to formulae
  - The forward-backward algorithm remains unchanged
  - The computation of gammas remains unchanged
  - The estimation of the parameters of state output distributions remains unchanged
- Transition probability computations also remain unchanged
  - Self-transition probability $T_{ii} = 0$ for Null states and this doesn't change
- NULL states have no observations associated with them; hence no state output densities need be learned for them

Learning From Word Sequences

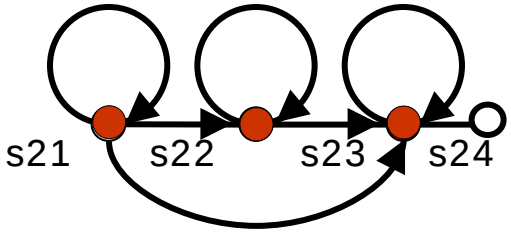
- In the explanation so far we have seen how to deal with a *single* string of words
- But when we're learning from a set of word sequences, words may occur in any order
  - E.g. Training recording no. 1 may be "word1 word2" and recording 2 may be "word2 word1"
- Words may occur multiple times within a single recording
  - E.g "word1 word2 word3 word1 word2 word3"
- All instances of any word, regardless of its position in the sentence, must contribute towards learning the HMM for it
  - E.g. from recordings such as "word1 word2 word3 word2 word1" and "word3 word1 word3", we should learn models for word1, word2, word3 etc.

Learning Word Models from Connected Recordings

- Best explained using an illustration
  - HMM for word1



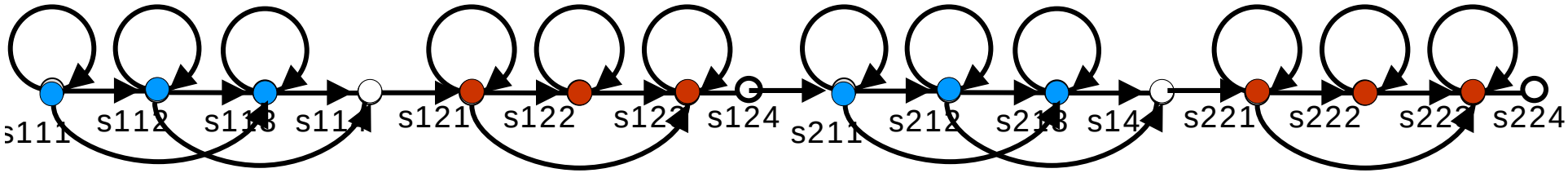
  - HMM for word 2



- Note *states* are labelled
  - E.g. state s11 is the 1st state of the HMM for word no. 1

Learning Word Models from Connected Recordings

- Model for “Word1 Word2 Word1 Word2”

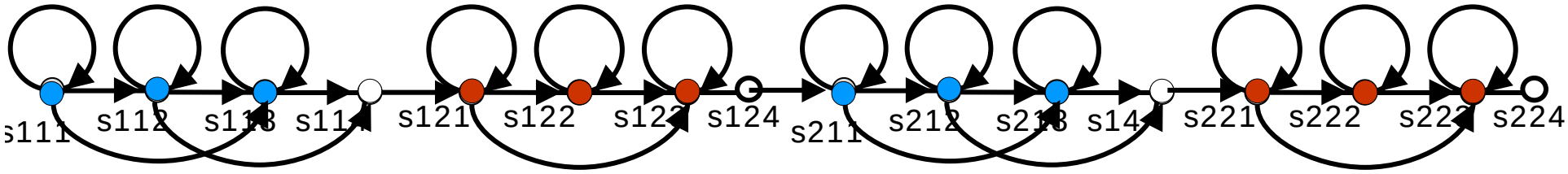


  - State indices are “sijk” referring to the k-th state of the j-th word in its i-th repetition
  - E.g. “s123” represents the third state of the 1st instance of word2

- If this were a single HMM we would have 16 states, a 16x16 transition matrix

Learning Word Models from Connected Recordings

- Model for “Word1 Word2 Word1 Word2



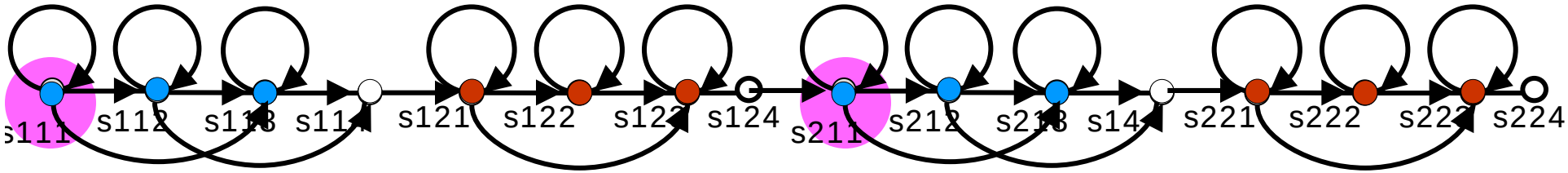
- The update formula would be as below
  - Only state output distribution parameter formulae are shown. It is assumed that the distributions are Gaussian. But the generalization to other formuale is straight-forward

$$\mu_{s_{i,j,k}} = \frac{\sum_u \sum_t \gamma_{s_{i,j,k},u,t} x_{u,t}}{\sum_u \sum_t \gamma_{s_{i,j,k},u,t}}$$

$$C_{s_{i,j,k}} = \frac{\sum_u \sum_t \gamma_{s_{i,j,k},u,t} (x - \mu_{s_{i,j,k}})^T (x - \mu_{s_{i,j,k}})}{\sum_u \sum_t \gamma_{s_{i,j,k},u,t}}$$

Combining Word Instances

- Model for "Word1 Word2 Word1 Word2

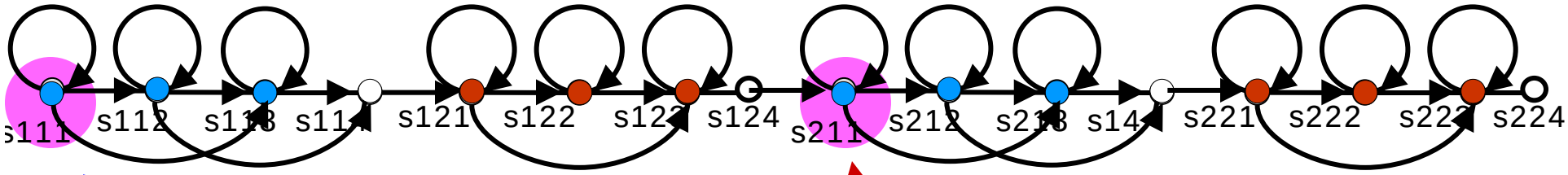


- However, these states are the same!
  - Data at either of these states are from the first state of word 1
  - This leads to the following modification for the parameters of s11 (first state of word1)

$$\mu_{s_{1,1}} = \frac{\sum_u \sum_t \gamma_{s_{1,1,1},u,t} x_{u,t} + \sum_u \sum_t \gamma_{s_{2,1,1},u,t} x_{u,t}}{\sum_u \sum_t \gamma_{s_{1,1,1},u,t} + \sum_u \sum_t \gamma_{s_{2,1,1},u,t}}$$

Combining Word Instances

- Model for "Word1 Word2 Word1 Word2



- However, these states are the same!
  - Data at either of these states are from the first state of word 1
  - This leads to the following modification for the parameters of s11 (first state of word1)

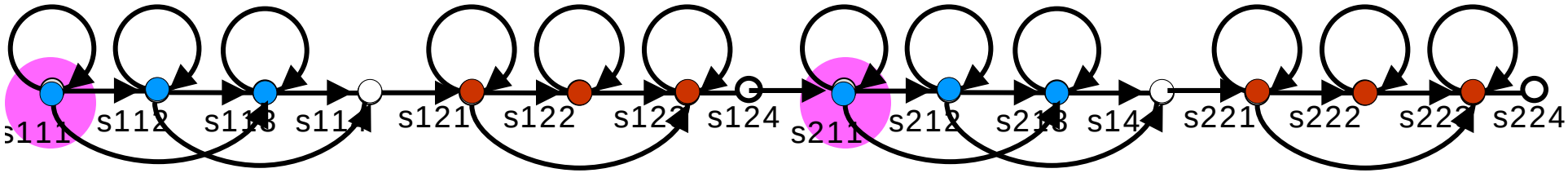
Formula for Mean

$$\mu_{s_{1,1}} = \frac{\sum_u \sum_t \gamma_{s_{1,1,1},u,t} x_{u,t} + \sum_u \sum_t \gamma_{s_{2,1,1},u,t} x_{u,t}}{\sum_u \sum_t \gamma_{s_{1,1,1},u,t} + \sum_u \sum_t \gamma_{s_{2,1,1},u,t}}$$

NOTE: Both terms From both instances of the word are being combined

Combining Word Instances

- Model for "Word1 Word2 Word1 Word2



$$C_{s_{1,1}} = \frac{\sum_u \sum_t \gamma_{s_{1,1,1},u,t} (x-\mu_{s_{1,1}})^T (x-\mu_{s_{1,1}}) + \sum_u \sum_t \gamma_{s_{2,1,1},u,t} (x-\mu_{s_{1,1}})^T (x-\mu_{s_{1,1}})}{\sum_u \sum_t \gamma_{s_{1,1,1},u,t} + \sum_u \sum_t \gamma_{s_{2,1,1},u,t}}$$

Formula for variance

Note, this is the mean of s11 (not s111 or s211)

Combining Word Instances

- The parameters of all states of all words are similarly computed
  - The principle extends easily to large corpora with many word recordings
- The HMM training formulae may be generally rewritten as:
  - Formlae are for parameters of Gaussian state output distributions
  - Transition probability updates rules are not shown, but are similar
  - Extensions to GMMs are straight forward

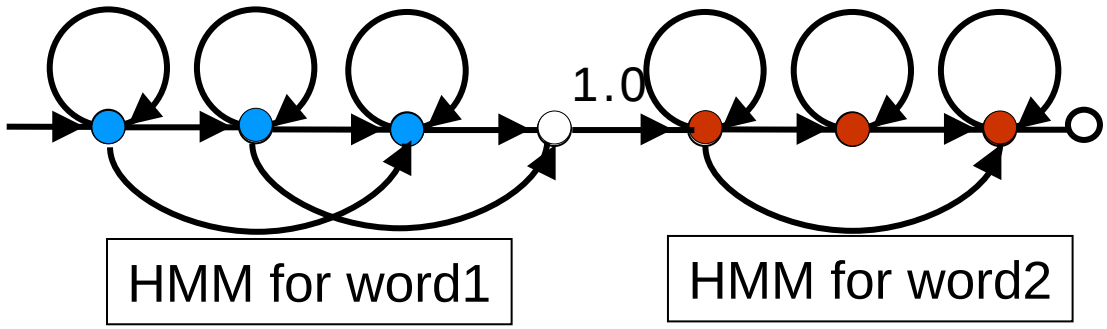
$$\mu_{s_{j,k}} = \frac{\sum_u \sum_i \sum_t \gamma_{s_{i,j,k},u,t} x_{u,t}}{\sum_u \sum_i \sum_t \gamma_{s_{i,j,k},u,t}}$$

$$C_{s_{j,k}} = \frac{\sum_u \sum_i \sum_t \gamma_{s_{i,j,k},u,t} (x - \mu_{s_{j,k}})^T (x - \mu_{s_{j,k}})}{\sum_u \sum_i \sum_t \gamma_{s_{i,j,k},u,t}}$$

Summation over instances

Concatenating Word Models: Silences

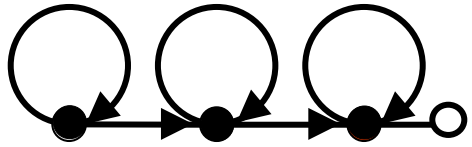
- People do not speak words continuously
- Often they pause between words
- If the recording was <word1> <pause> <word2> the following model would be inappropriate



- The above structure does *not* model the pause between the words
  - It only permits direct transition from word1 to word2
- The <pause> must be incorporated somehow

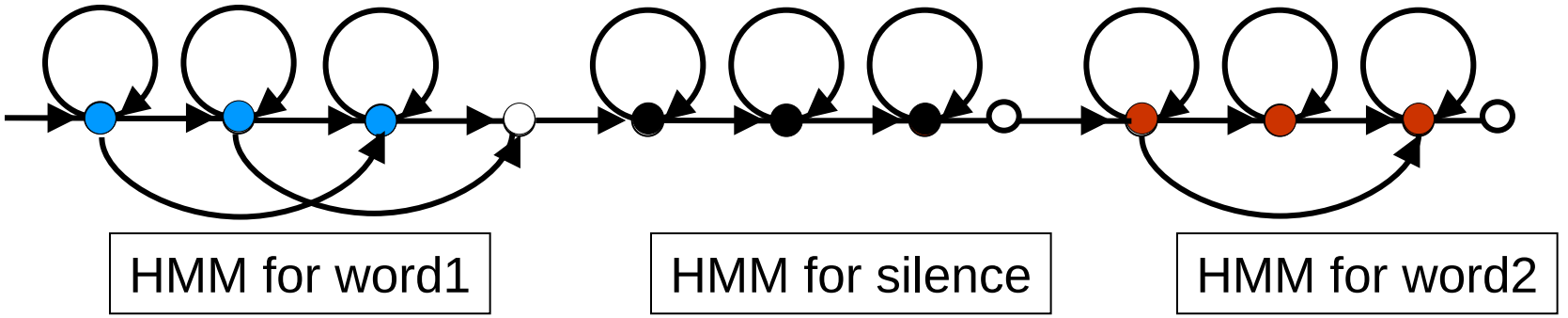
Pauses are Silences

- Silences have spectral characteristics too
  - A sequence of low-energy data
  - Usually represents the background signal in the recording conditions
- We build an HMM to represent silences



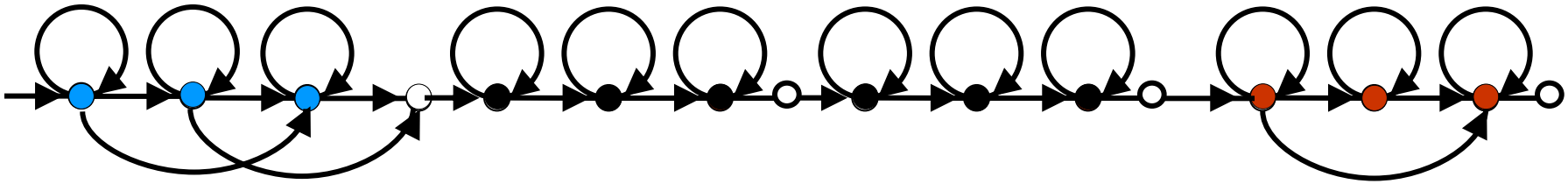
Incorporating Pauses

- The HMM for <word1> <pause> <word2> is easy to build now



Incorporating Pauses

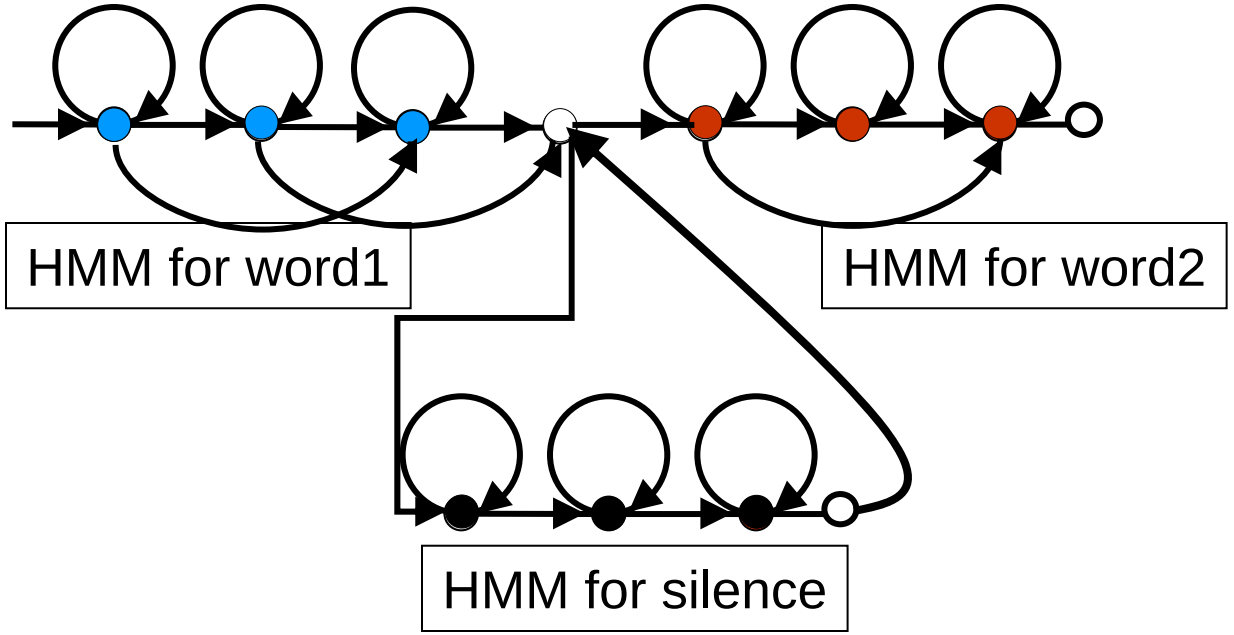
- If we have a long pause: Insert multiple pause models



HMM for word1 | HMM for silence | HMM for silence | HMM for word2

Incorporating Pauses

- What if we do not know how long the pause is

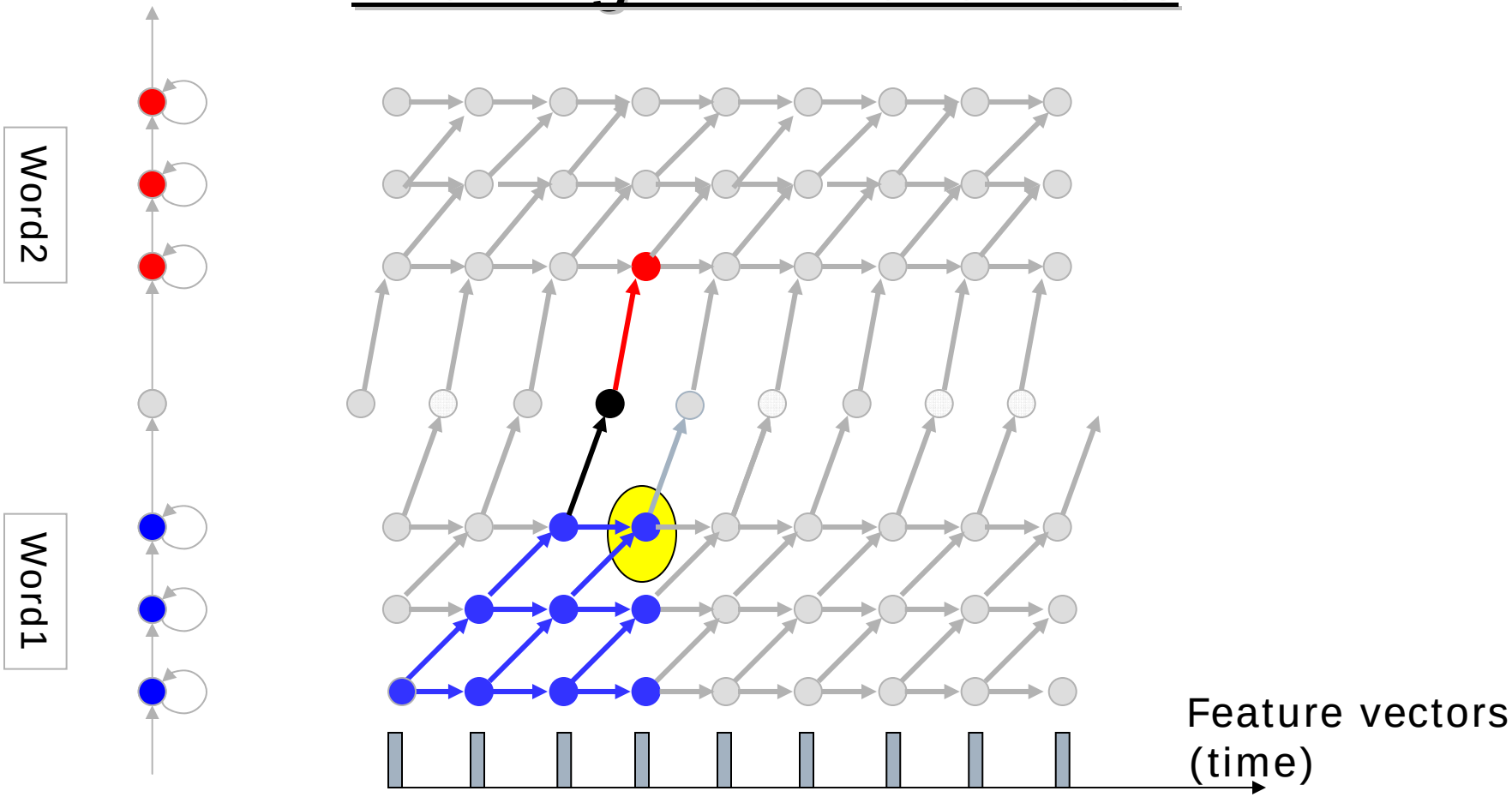


- We allow the pause to be *optional*
  - There is a transition from word1 to word2
  - There is *also* a transition from word1 to silence
  - Silence loops back to the junction of word1 and word2
  - This allows for an *arbitrary* number of silences to be inserted

Another Implementational Issue: Complexity

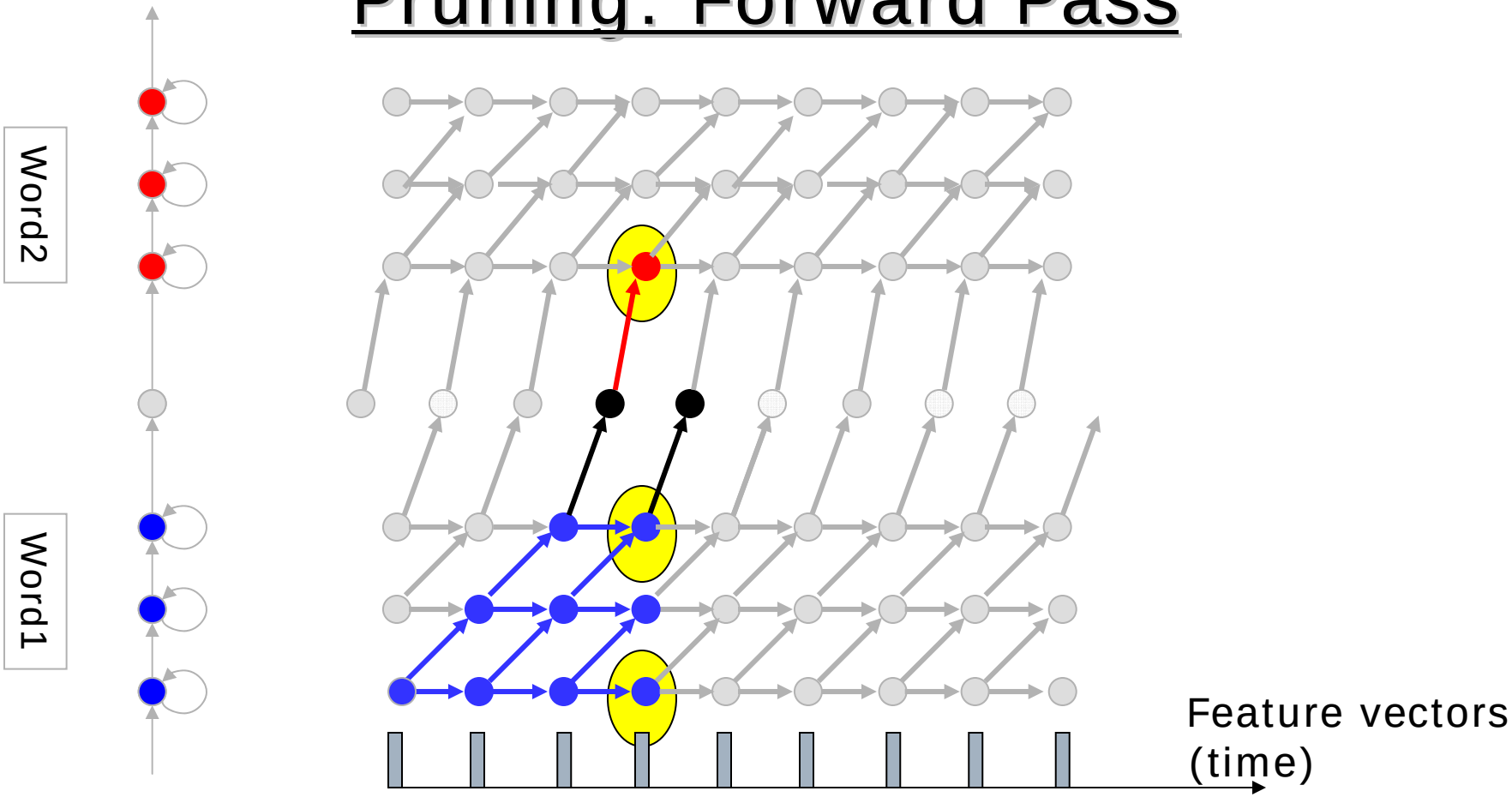
- Long utterances with many words will have many states
- The size of the trellis grows as NT, where N is the no. of states in the HMM and T is the length of the observation sequence
- N in turn increases with T and is roughly proportional to T
  - Longer utterances have more words
- The computational complexity for computing alphas, betas, or the best state sequence is $O(N^2T)$
- Since N is proportional to T, this becomes $O(T^3)$
- This number can be very large
  - The computation of the forward algorithm could take forever
  - So also for the forward algorithm

Pruning: Forward Pass



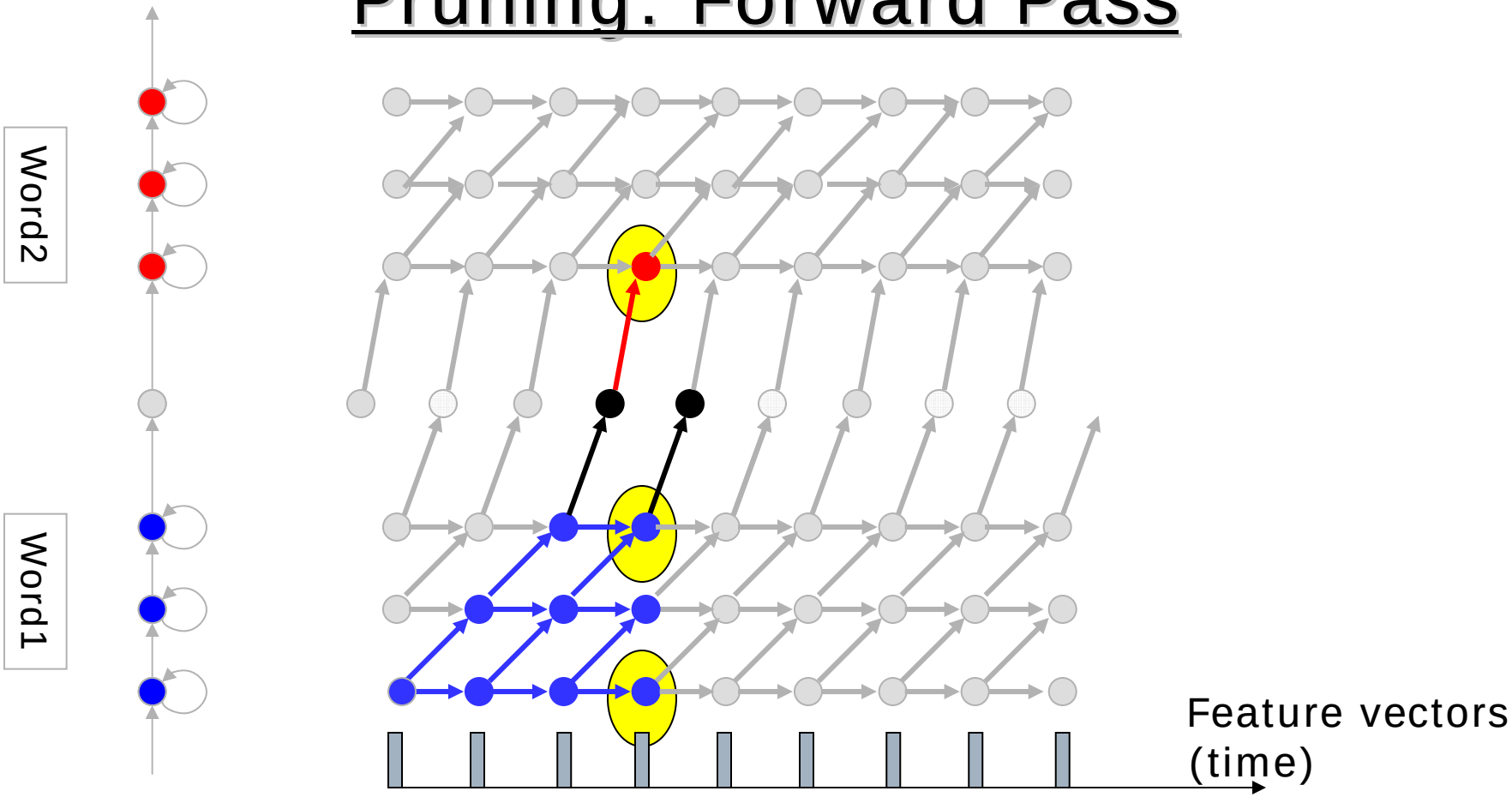
- In the forward pass, at each time find the best scoring state

Pruning: Forward Pass



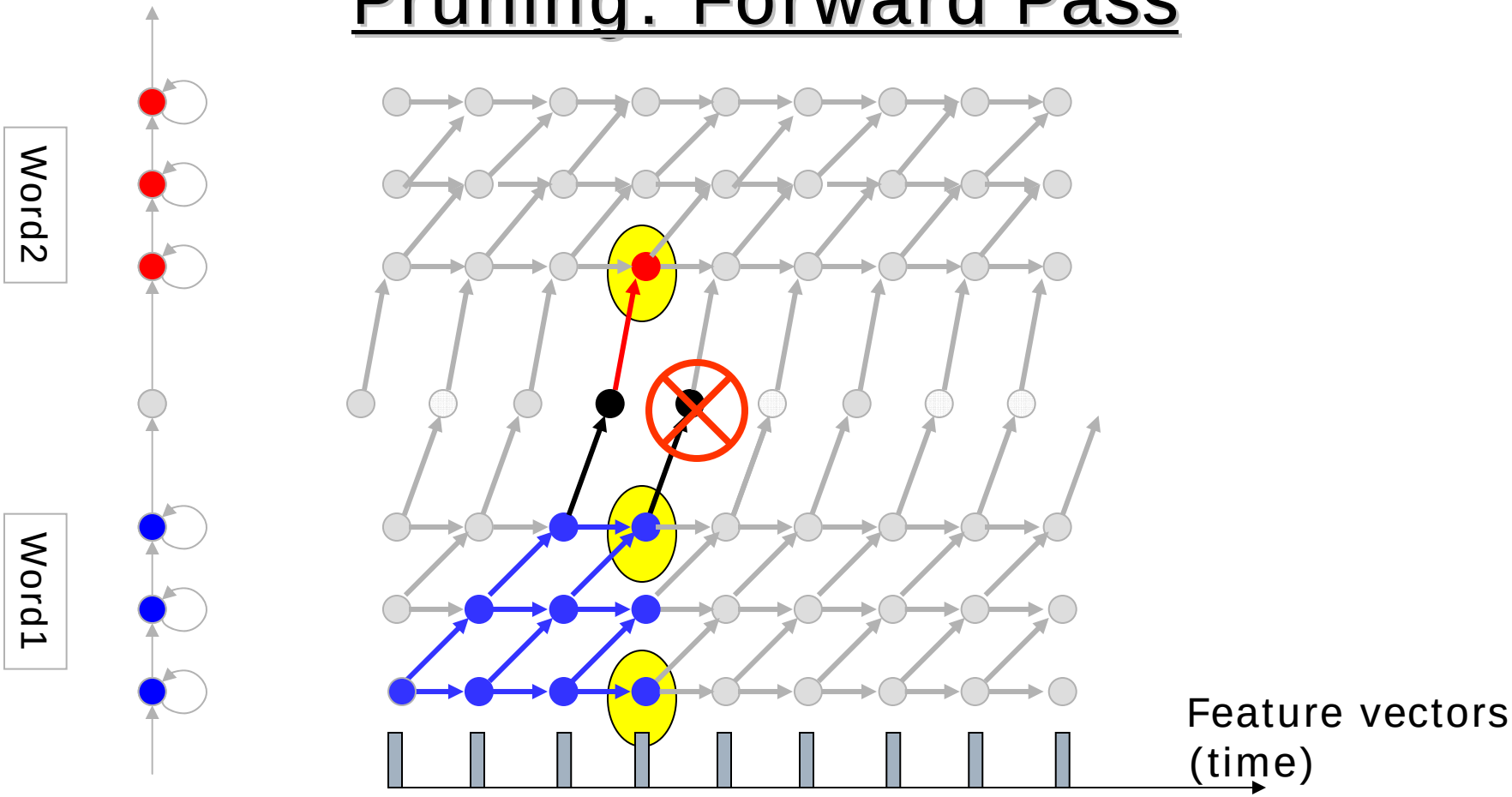
- In the forward pass, at each time find the best scoring state
- Retain all states with a score > k*bestscore
  - k is known as the beam
- States with scores less than this beam are not considered in the next time instant

Pruning: Forward Pass



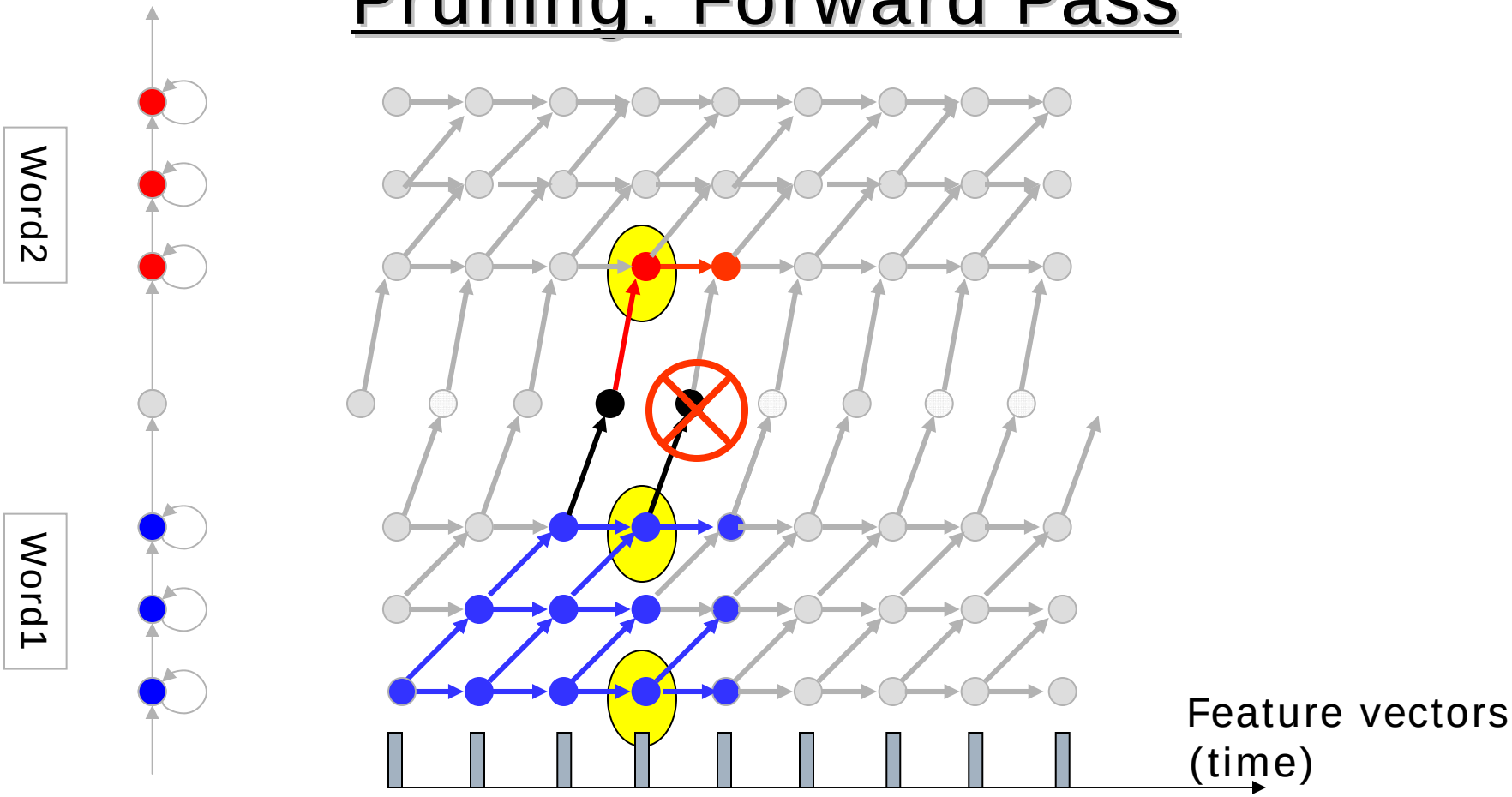
- The rest of the states are assumed to have zero probability
  - I.e. they are pruned
- Only the selected states carry forward
  - First to NON EMITTING states

Pruning: Forward Pass



- The rest of the states are assumed to have zero probability
  - I.e. they are pruned
- Only the selected states carry forward
  - First to NON EMITTING states – which may also be pruned out after comparison to other non-emitting states in the same column

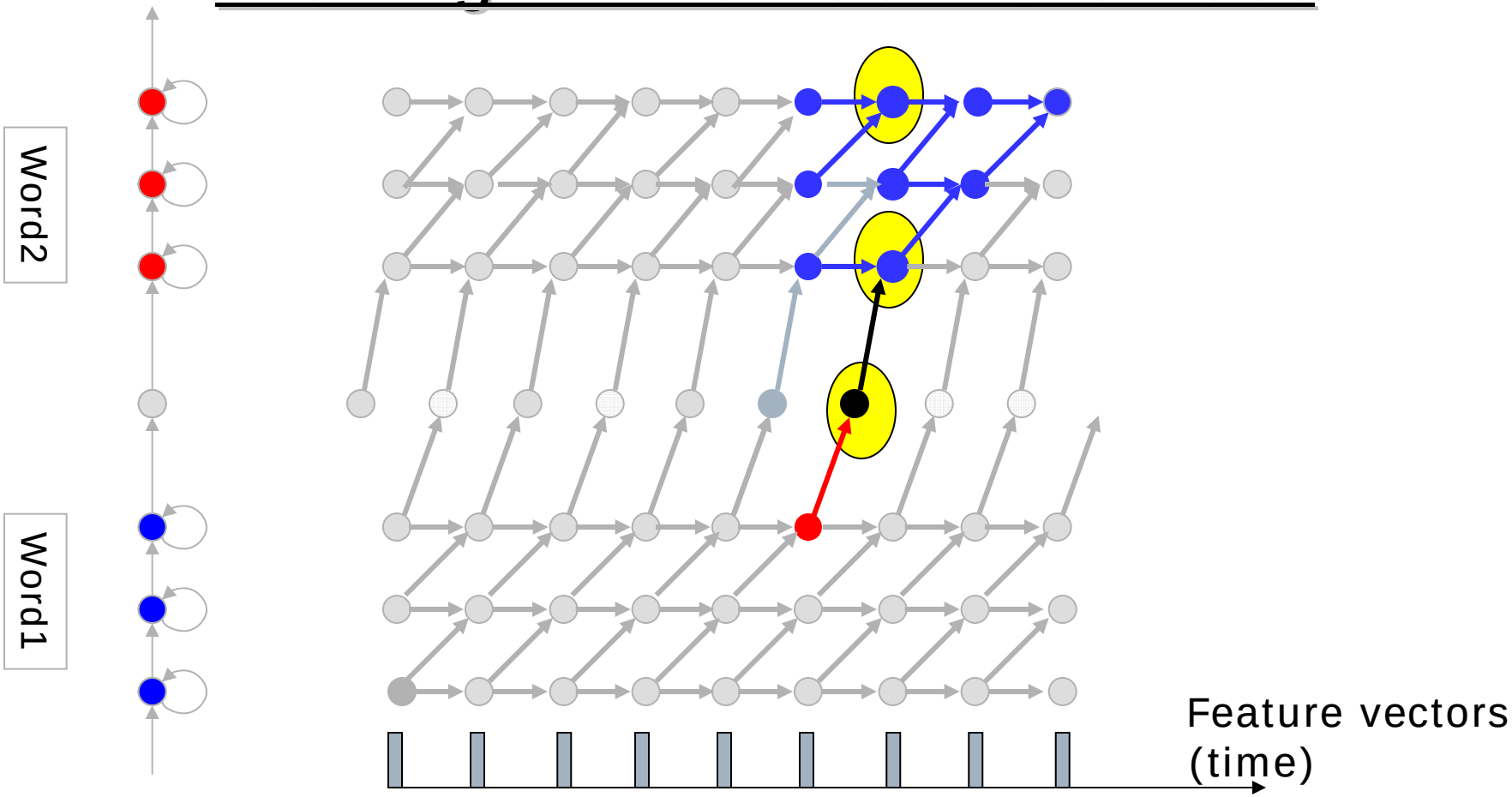
Pruning: Forward Pass



- The rest are carried forward to the next time

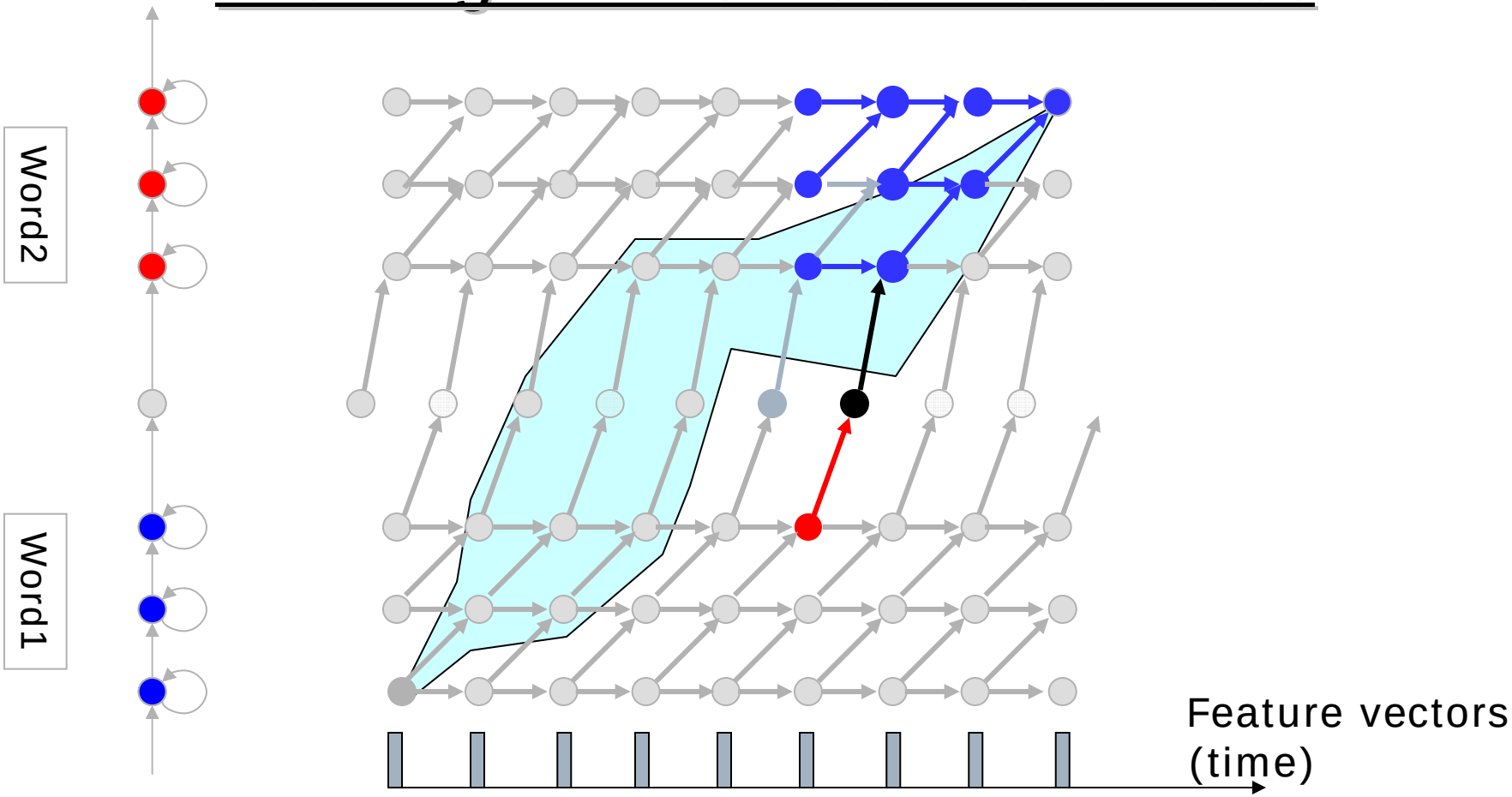
t

Pruning In the Backward Pass



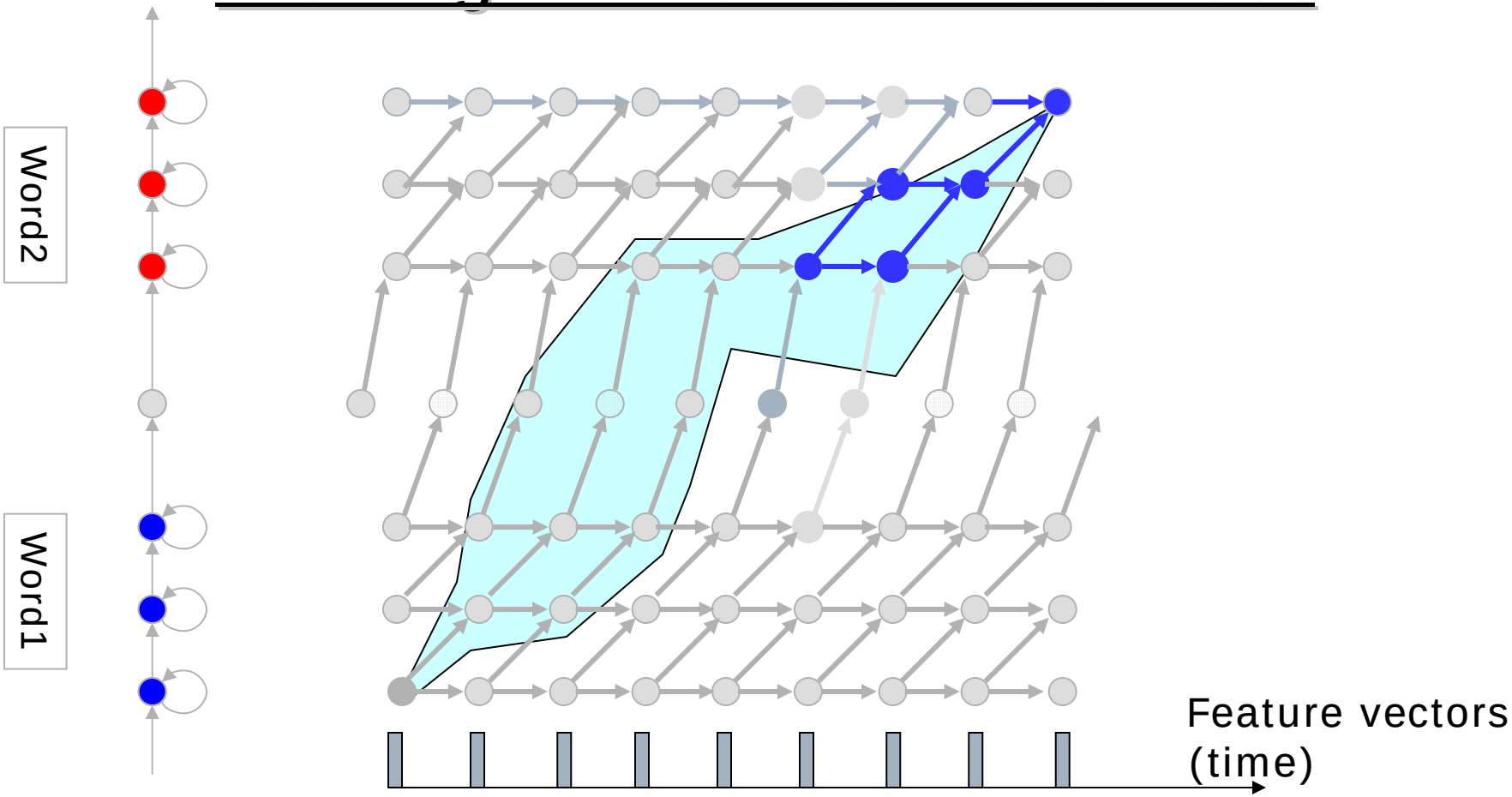
- A similar Heuristic may be applied in the backward pass for speedup
  - But this can be inefficient

Pruning In the Backward Pass



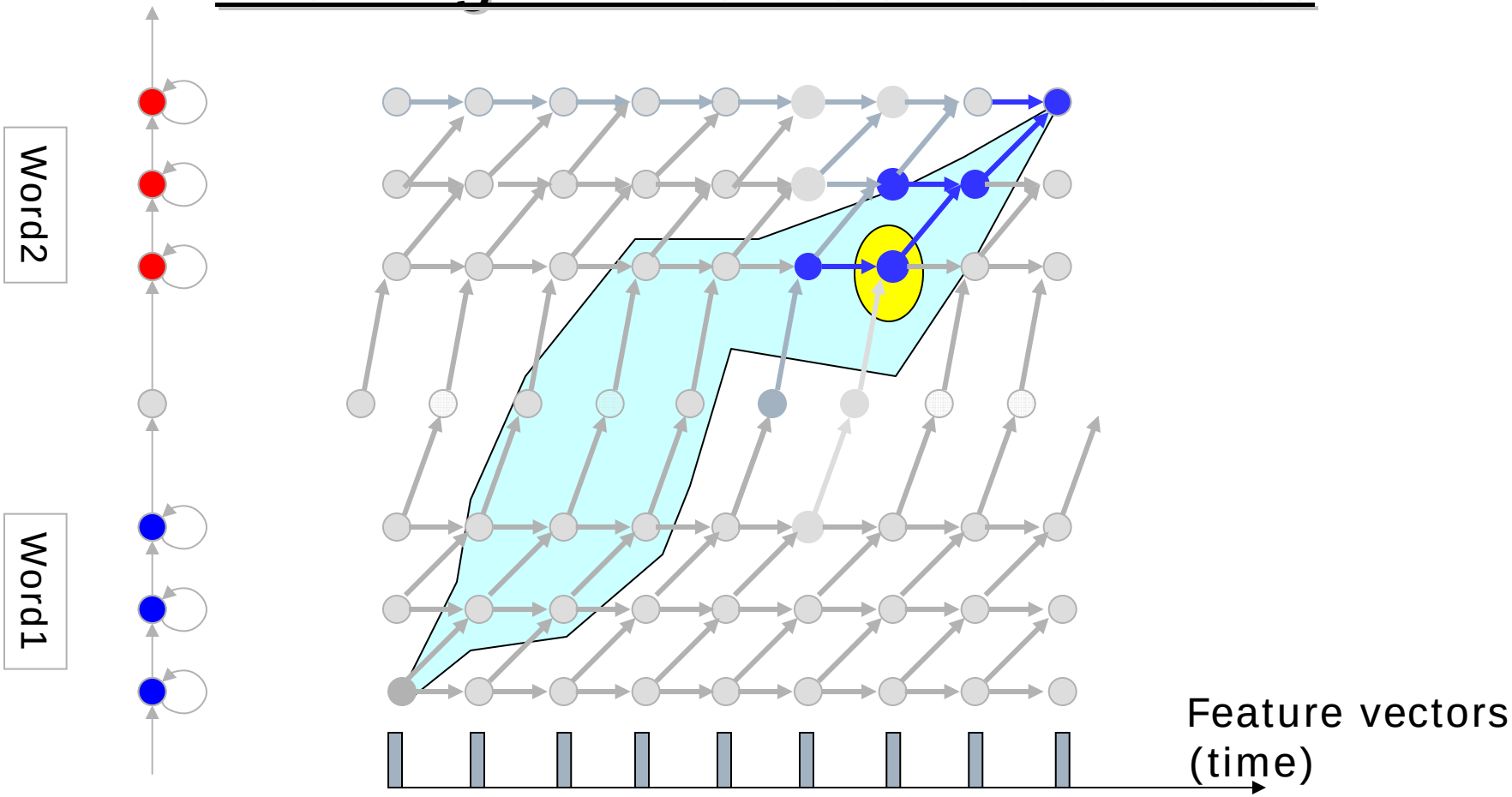
- The forward pass has already pruned out much of the trellis
  - This region of the trellis has 0 probability and need not be considered

Pruning In the Backward Pass



- The forward pass has already pruned out much of the trellis
  - This region of the trellis has 0 probability and need not be considered
  - The backward pass only needs to evaluate paths within this portion

Pruning In the Backward Pass



- The forward pass has already pruned out much of the trellis
  - This region of the trellis has 0 probability and need not be considered
  - The backward pass only needs to evaluate paths within this portion
    - Pruning may still be performed going backwards

Words are not good units for recognition

- For all but the smallest tasks words are not good units
- For example, to recognize speech of the kind that is used in broadcast news, we would need models for all words that may be used
  - This could exceed 100000 words
- As we will see, this quickly leads to problems

The problem with word models

- Word model based recognition:
  - Obtain a “template” or “model” for every word you want to recognize
    - And maybe for garbage
  - Recognize any given input data as being one of the ***known*** words

- Problem: We need to train models for every word we wish to recognize
  - E.g., if we have trained models for words “zero, one, .. nine”, and wish to add “oh” to the set, we must now learn a model for “oh”
  - Inflexible

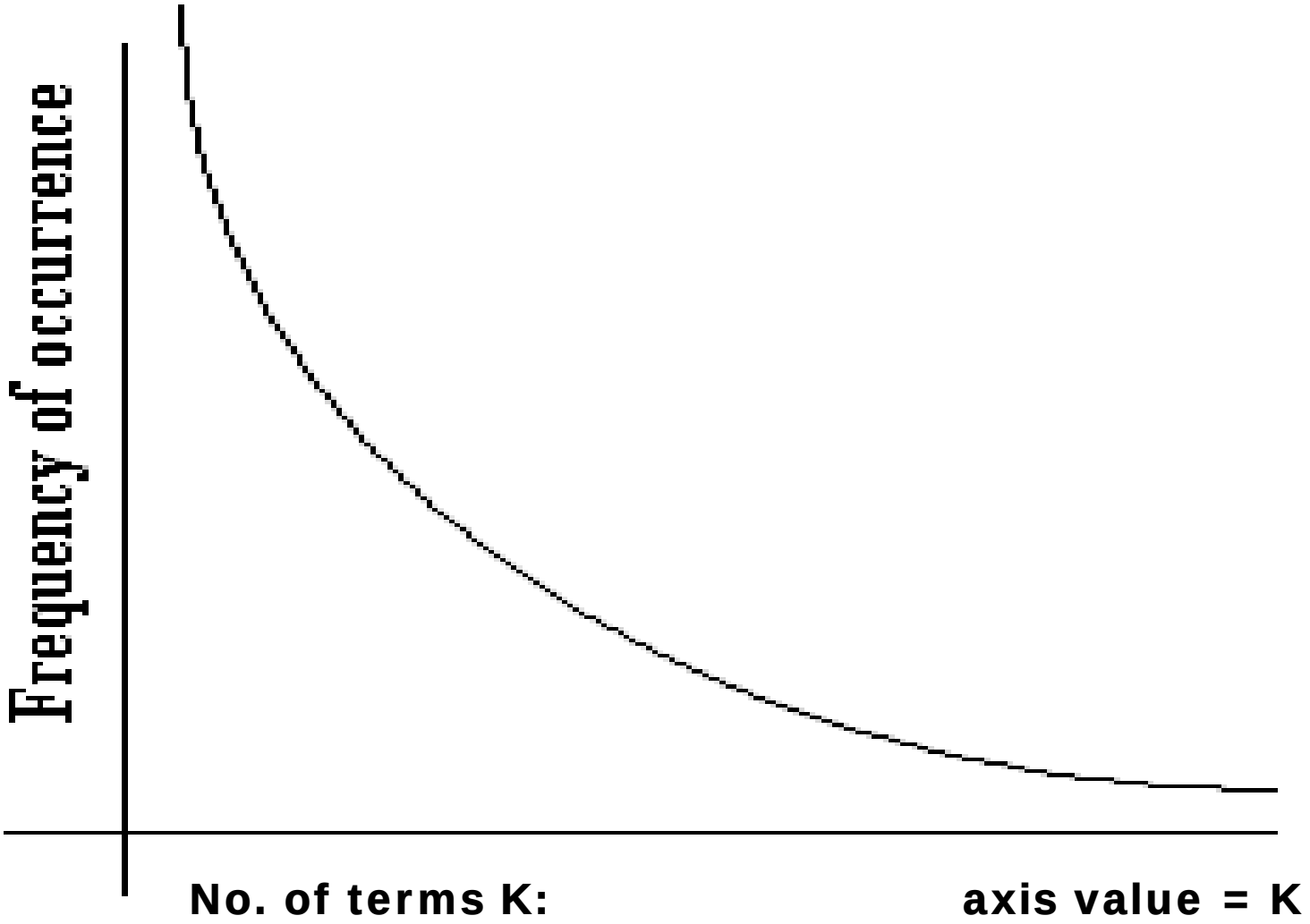
- ***Training needs data***
  - We can only learn models for words for which we have training data available

Zipf's Law

- Zipf's law: The number of events that occur often is small, but the number of events that occur very rarely is very large.
  - E.g. you see a lot of dogs every day. There is one species of animal you see very often.
  - There are thousands of species of other animals you don't see except in a zoo. i.e. there are a very large number of species which you don't see often.

- If n represents the number of times an event occurs in a unit interval, the number of events that occur n times per unit time is proportional to $1/n^{\alpha}$, where α is greater than 1
  - George Kingsley Zipf originally postulated that $\alpha = 1$.
    - Later studies have shown that α is $1 + \varepsilon$, where ε is slightly greater than 0

Zipf's Law



Zipf's Law also applies to Speech and Text

- The following are examples of the most frequent and the least frequent words in 1.5 million words of broadcast news representing 70 of hours of speech
  - THE: 81900
  - AND: 38000
  - A: 34200
  - TO: 31900
  - ..
  - ADVIL: 1
  - ZOOLOGY: 1
- Some words occur more than 10000 times (very frequent)
  - There are only a few such words: 16 in all
- Others occur only once or twice – 14900 words in all
  - Almost 50% of the vocabulary of this corpus
- The variation in number follows Zipf's law: there are a small number of frequent words, and a very large number of rare words
  - Unfortunately, the rare words are often the most important ones – the ones that carry the most information

Word models for Large Vocabularies

- If we trained HMMs for individual words, most words would be trained on a small number (1-2) of instances (Zipf's law strikes again)
  - The HMMs for these words would be poorly trained
  - The problem becomes more serious as the vocabulary size increases
- No HMMs can be trained for words that are never seen in the training corpus
- Direct training of word models is not an effective approach for large vocabulary speech recognition

Sub-word Units

- Observation: Words in any language are formed by sequentially uttering a set of sounds
- The set of these sounds is small for any language
- Any word in the language can be defined in terms of these units
- The most common sub-word units are “phonemes”
  - The technical definition of “phoneme” is obscure
  - For purposes of speech recognition, it is a small, repeatable unit with consistent internal structure.
    - Although usually defined with linguistic motivation

Examples of Phonemes

- AA: As in F AA ST
- AE: As in B AE T M AE N
- AH: As in H AH M (HUM)
- B: As in B EAST

- Etc.

- Words in the language are expressible (in their spoken form) in terms of these phonemes

Phonemes and Pronunciation Dictionaries

- To use Phonemes as sound units, the mapping from words to phoneme sequences must be specified
  - Usually specified through a mapping table called a *dictionary*

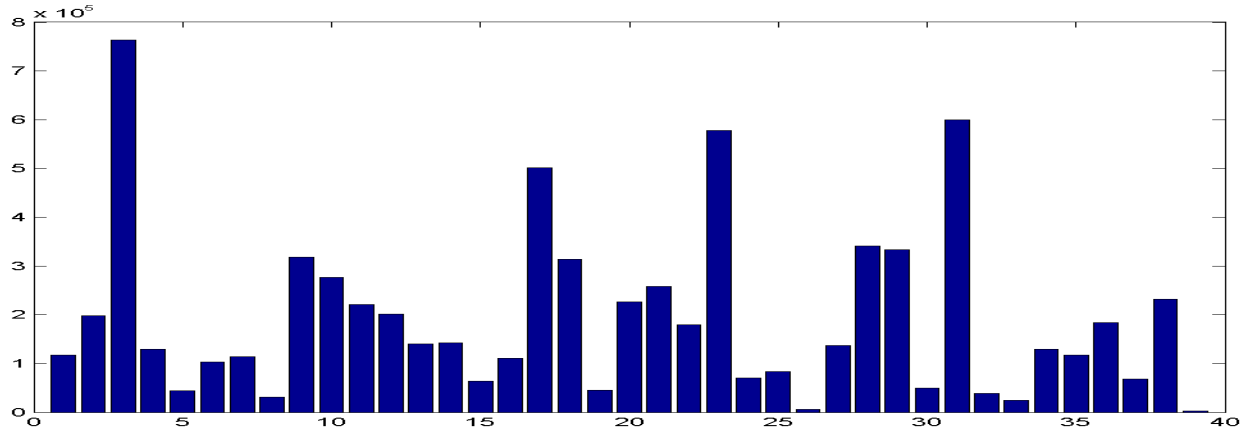
Mapping table (dictionary)

Eight	ey t
Four	f ow r
One	w ax n
Zero	z iy r ow
Five	f ay v
Seven	s eh v ax n

- Every word in the training corpus is converted to a sequence of phonemes
  - The transcripts for the training data effectively become sequences of phonemes
- HMMs are trained for the phonemes

Beating Zipf's Law

- Distribution of phonemes in the BN corpus



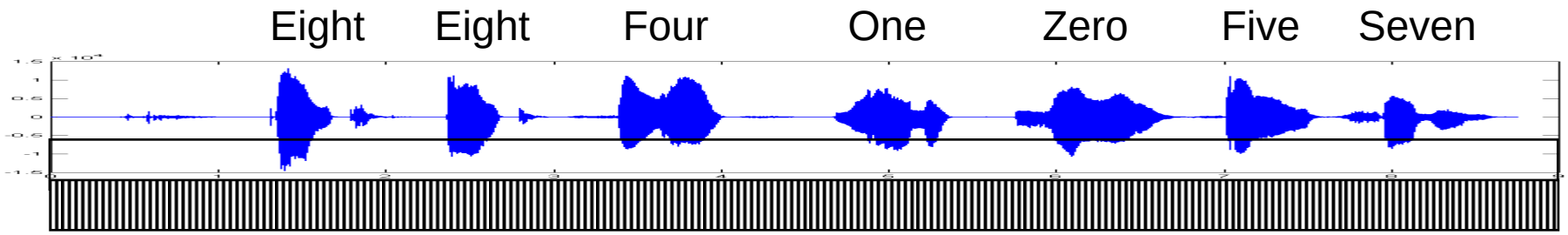
Histogram of the number of occurrences of the 39 phonemes in 1.5 million words of Broadcast News

- There are far fewer "rare" phonemes, than words
  - This happens because the probability mass is distributed among fewer unique events
- If we train HMMs for phonemes instead of words, we will have enough data to train all HMMs

But we want to recognize ***Words***

- Recognition will *still* be performed over words
- The HMMs for words are constructed by concatenating the HMMs for the individual phonemes within the word
  - In order provided by the dictionary
  - Since the component phoneme HMMs are well trained, the constructed word HMMs will also be well trained, even if the words are very rare in the training data
- This procedure has the advantage that we can now create word HMMs for words that were *never seen* in the acoustic model training data
  - We only need to know their pronunciation
  - Even the HMMs for these unseen (new) words will be well trained

Word-based Recognition

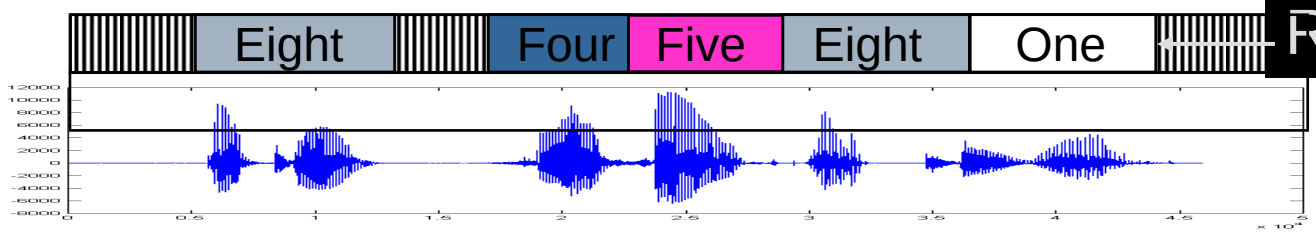


Trainer
Learns characteristics of sound units

Decoder
Identifies sound units based on learned characteristics

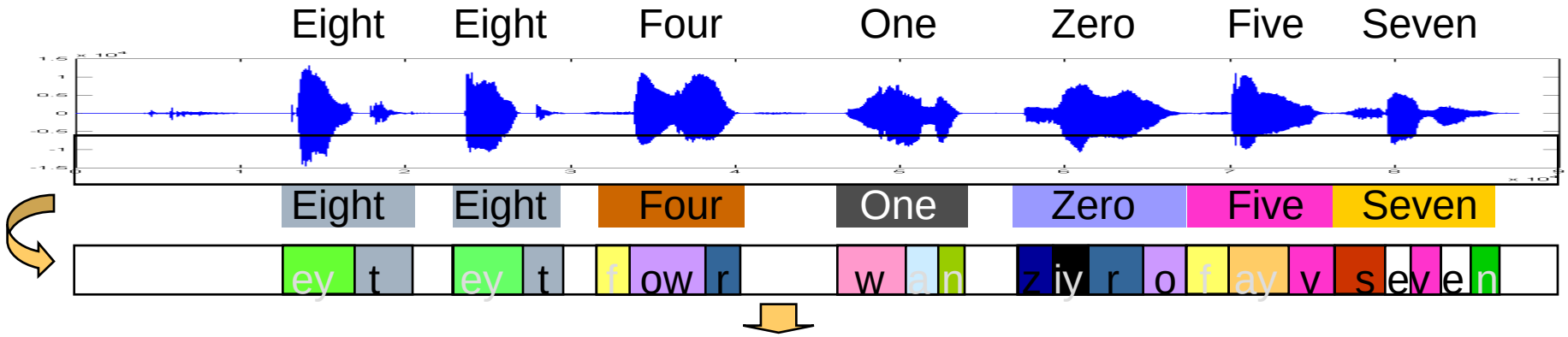
Word as unit

Insufficient data to train every word. Words not seen in training not recognized



Enter Four Five Eight Two One ← Spoken

Phoneme based recognition



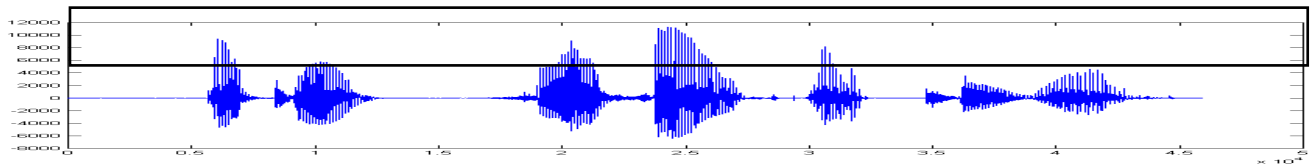
Dictionary

Eight: ey t
Four: f ow r
One: w a n
Zero: z iy r ow
Five: f ay v
Seven: s e v e n

Trainer
Learns characteristics of sound units

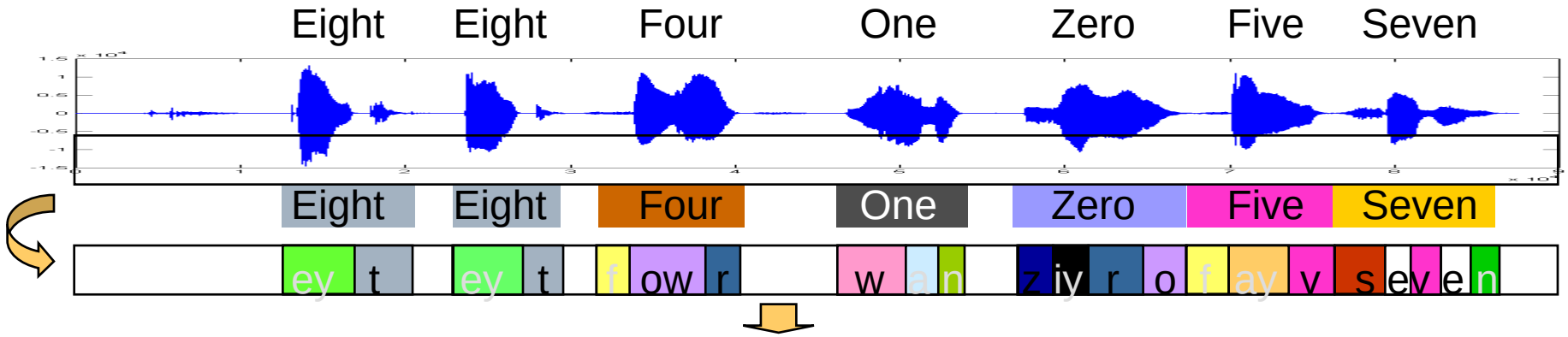
Map words into phoneme sequences

Decoder
Identifies sound units based on learned characteristics



Enter Four Five Eight Two One

Phoneme based recognition



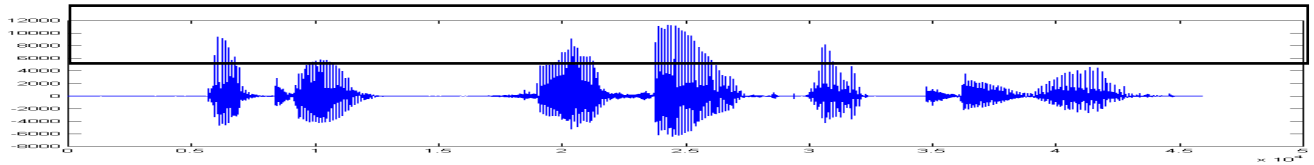
Dictionary

Eight: ey t
Four: f ow r
One: w a n
Zero: z iy r ow
Five: f ay v
Seven: s e v e n
Enter: e n t e r
two: t uw

Trainer
Learns characteristics of sound units

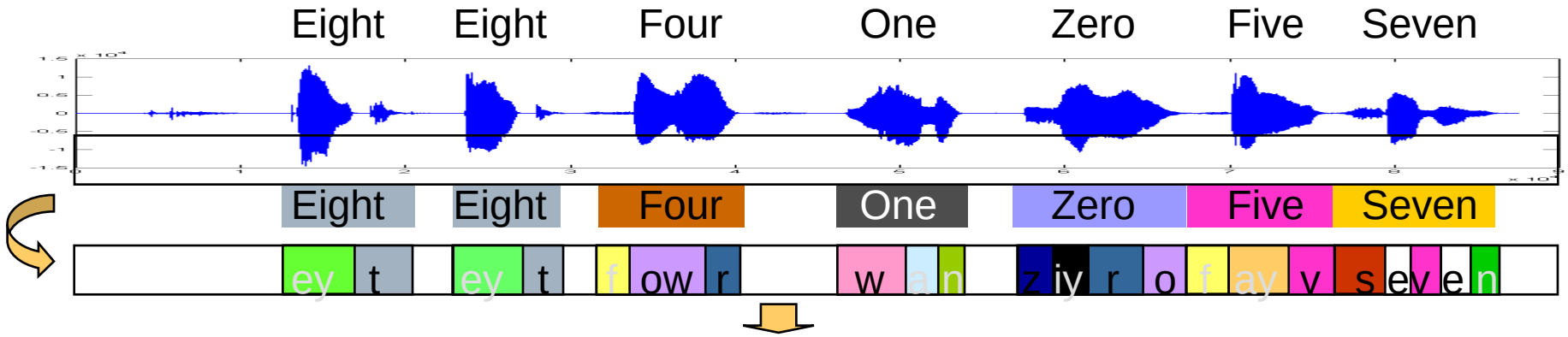
Decoder
Identifies sound units based on learned characteristics

Map words into phoneme sequences and learn models for phonemes
New words can be added to the dictionary



Enter Four Five Eight Two One

Phoneme based recognition



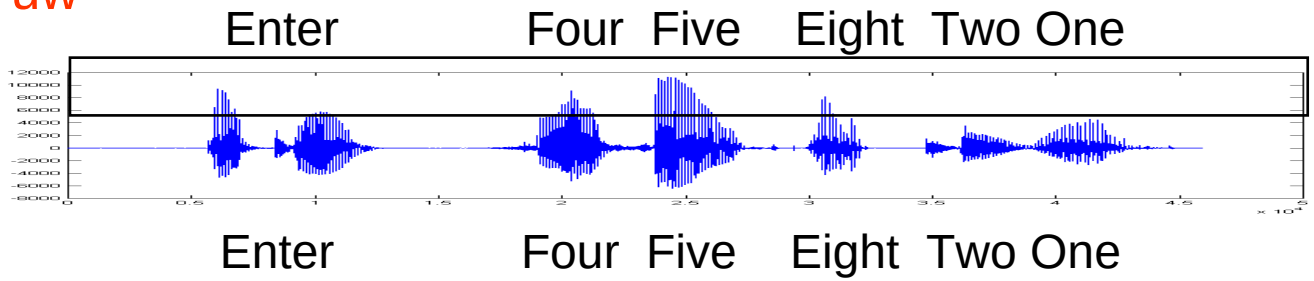
Dictionary

Eight: ey t
Four: f ow r
One: w a n
Zero: z iy r ow
Five: f ay v
Seven: s e v e n
Enter: e n t e r
two: t uw

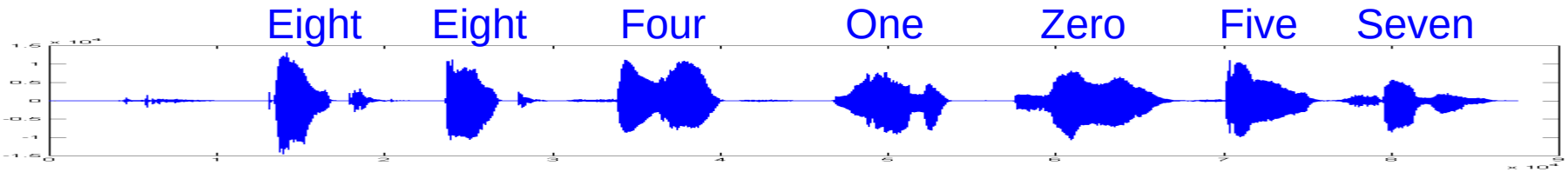
Trainer
Learns characteristics of sound units

Decoder
Identifies sound units based on learned characteristics

Map words into phoneme sequences and learn models for phonemes
New words can be added to the dictionary AND RECOGNIZED



Words vs. Phonemes



Unit = whole word

Average training examples per unit = 7/6 =~ 1.17

ey t ey t f ow r w a n iy r ow f ay v s e v e n

Unit = sub-word

Average training examples per unit = 22/14 =~ 1.57

More training examples = better statistical estimates of model (HMM) parameters

The difference between training instances/unit for phonemes and words increases dramatically as the training data and vocabulary increase

How do we define phonemes?

- The choice of phoneme set is not obvious
  - Many different variants even for English

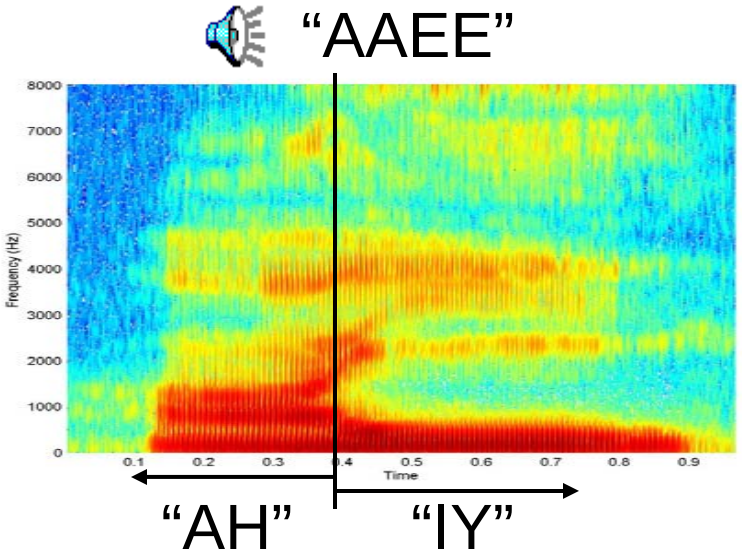
- Phonemes should be different from one another, otherwise training data can get diluted
  - Consider the following (hypothetical) example:
  - Two phonemes “AX” and “AH” that sound nearly the same
    - If during training we observed 5 instances of “AX” and 5 of “AH”
    - There might be insufficient data to train either of them properly
    - However, if both sounds were represented by a common symbol “A”, we would have 10 training instances!

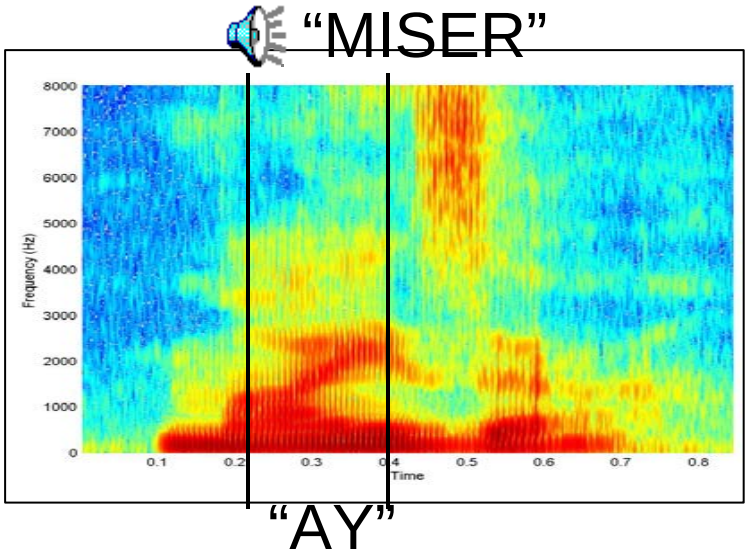
Defining Phonemes

- They should be significantly different from one another to avoid inconsistent labelling
  - E.g. “AX” and “AH” are similar but not identical
  - ONE: W AH N
    - AH is clearly spoken
  - BUTTER: B AH T AX R
    - The AH in BUTTER is sometimes spoken as AH (clearly enunciated), and at other times it is very short “B AX T AX R”
    - The entire range of pronunciations from “AX” to “AH” may be observed
      - Not possible to make clear distinctions between instances of B AX T and B AH T
    - Training on many instances of BUTTER can result in AH models that are very close to that of AX!
      - Corrupting the model for ONE!

Defining a Phoneme

- Other inconsistencies are possible
  - Diphthongs are sounds that begin as one vowel and end as another, e.g. the sound “AY” in “MY”
  - Must diphthongs be treated as pairs of vowels or as a single unit?
  - An example

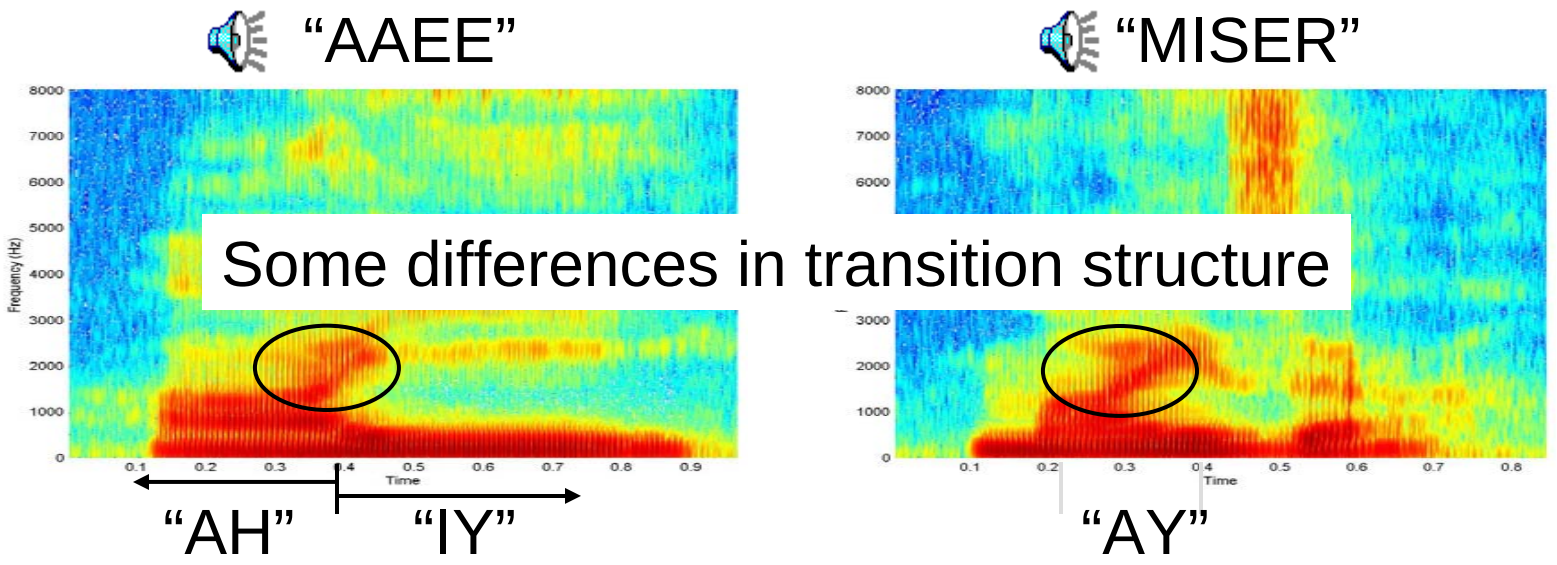




  - Is the sound in Miser the sequence of sounds “AH IY”, or is it the diphthong “AY”

Defining a Phoneme

- Other inconsistencies are possible
  - Diphthongs are sounds that begin as one vowel and end as another, e.g. the sound “AY” in “MY”
  - Must diphthongs be treated as p of vowels or as a single unit?
  - An example



  - Is the sound in Miser the sequence of sounds “AH IY”, or is it the diphthong “AY”

A Rule of Thumb

- If compound sounds occur frequently and have smooth transitions from one phoneme to the other, the compound sound can be single sound
  - Diphthongs have a smooth transition from one phoneme to the next
    - Some languages like Spanish have no diphthongs – they are always sequences of phonemes occurring across syllable boundaries with no guaranteed smooth transitions between the two
- Diphthongs: AI, EY, OY (English), UA (French) etc.
  - Different languages have different sets of diphthongs
- Stop sounds have multiple components that go together
  - A closure, followed by burst, followed by frication (in most cases)
- Some languages have ***triphthongs***

Phoneme Sets

- Conventional Phoneme Set for English:
  - Vowels: AH, AX, AO, IH, IY, UH, UW etc.
  - Diphthongs: AI, EY, AW, OY, UA etc.
  - Nasals: N, M, NG
  - Stops: K, G, T, D, TH, DH, P, B
  - Fricatives and Affricates: F, HH, CH, JH, S, Z, ZH etc.
- Different groups tend to use a different set of phonemes
  - Varying in sizes between 39 and 50!
- For some languages, the set of sounds represented by alphabets in the script are a good set of phonemes

Consistency is important

- The phonemes must be used consistently in the dictionary
  - E.g. You distinguish between two phonemes “AX” and “IX”. The two are distinct sounds
  - When composing the dictionary the two are not used consistently
    - “AX” is sometimes used in place of “IX” and vice versa
  - You would be better off using a single phoneme (e.g. “IH”) instead of the two distinct, but inconsistently used ones

- ***Consistency of usage is key!***

Recognition with Phonemes

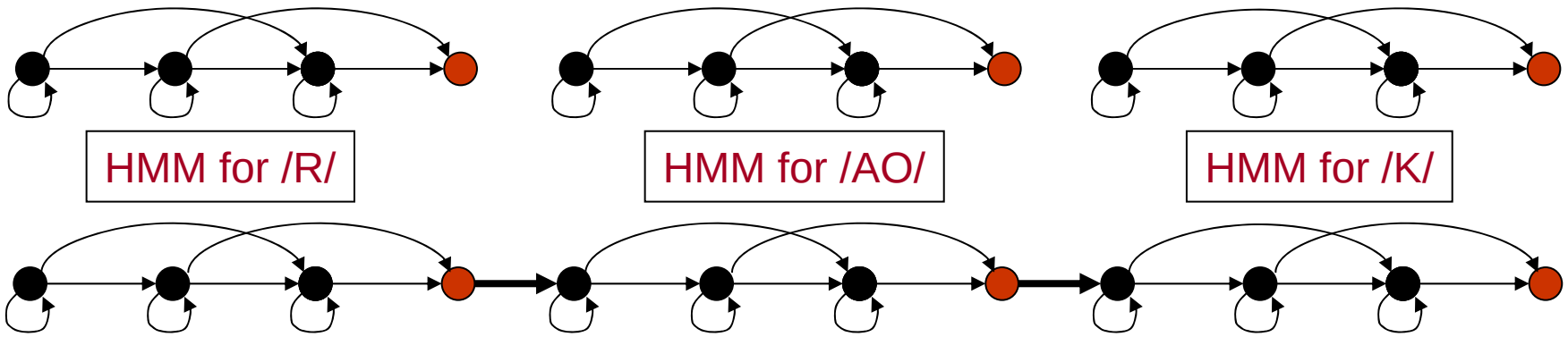
- The phonemes are only meant to enable better ***learning*** of templates
  - HMM or DTW models
- **We still recognize *words***
- The models for words are composed from the models for the subword units
- The HMMs for individual words are connected to form the Grammar HMM
- The best word sequence is found by Viterbi decoding
  - As we will see in a later lecture

Recognition with phonemes

Example:

Word	Phones
Rock	R AO K

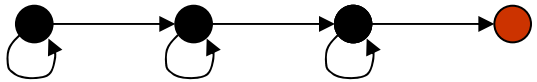
- Each phoneme is modeled by an HMM
- Word HMMs are constructed by concatenating HMMs of phonemes
- Composing word HMMs with phoneme units does not increase the complexity the grammar/language HMM



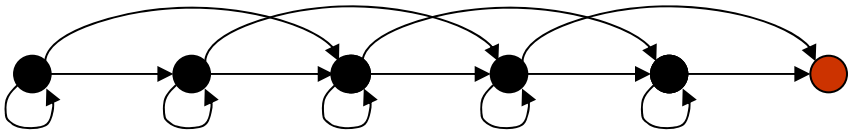
Composed HMM for ROCK

HMM Topology for Phonemes

- Most systems model Phonemes using a 3-state topology
  - All phonemes have the same topology



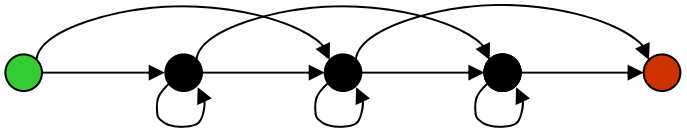
- Some older systems use a 5-state topology
  - Which permits states to be skipped entirely
  - This is not demonstrably superior to the 3-state topology



Composing a Word HMM

- Words are linear sequences of phonemes
- To form the HMM for a word, the HMMs for the phonemes must be linked into a larger HMM
- Two mechanisms:
  - Explicitly maintain a *non-emitting* state between the HMMs for the phonemes
    - Computationally efficient, but complicates time-synchronous search
  - Expand the links out to form a sequence of *emitting-only* states

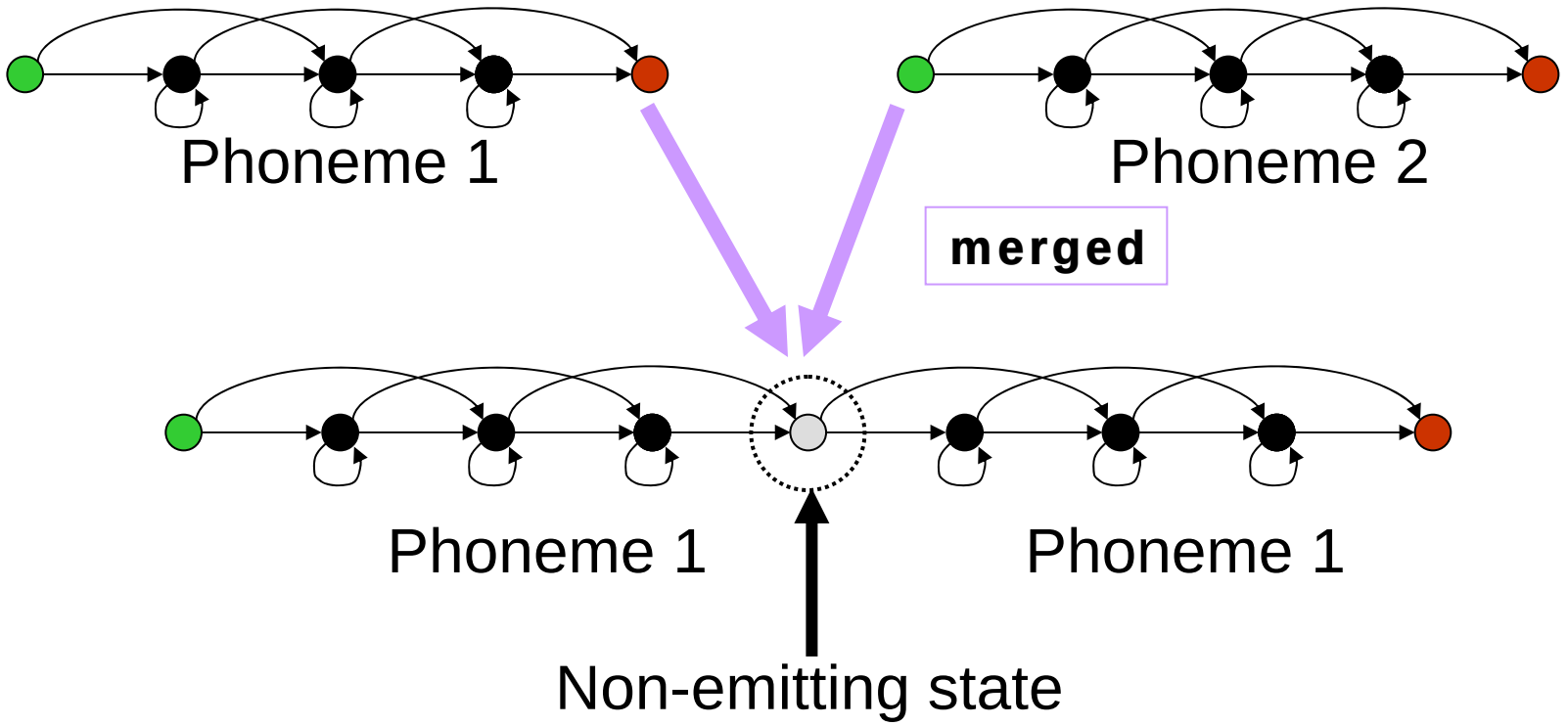
Generating and Absorbing States



- Phoneme HMMs are commonly defined with *two* non-emitting states
- One is a *generating* state that occurs at the beginning
  - All initial observations are assumed to be the outcome of transitions from this generating state
  - The *initial* state probability of any state is simply the transition probability from the generating state
- The absorbing state is a conventional non-emitting final state
- When phonemes are chained the absorbing state of one phoneme gets merged with the generating state of the next one

Linking Phonemes via Non-emitting State

- To link two phonemes, we create a new “non-emitting” state that represents both the absorbing state of the first phoneme and the generating state of the second phoneme



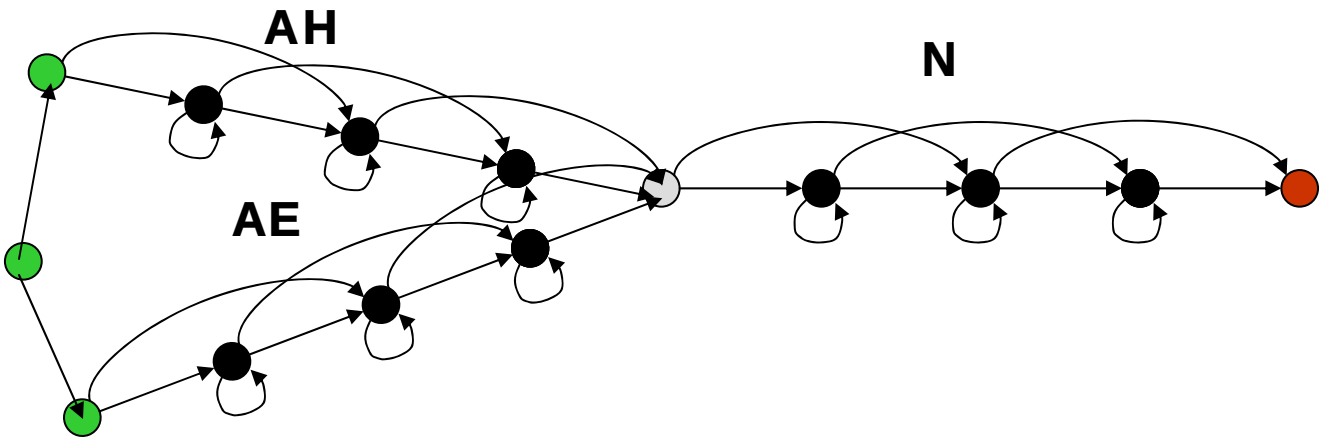
The problem of pronunciation

- There are often multiple ways of pronouncing a word.
  - Sometimes these pronunciation differences are semantically meaningful:
    - READ : R IY D (Did you read the book)
    - READ : R EH D (Yes I read the book)
  - At other times they are not
    - AN : AX N (That's an apple)
    - AN : AE N (An apple)

- These are typically identified in a dictionary through markers
  - READ(1) : R IY D
  - READ(2) : R EH D

Multiple Pronunciations

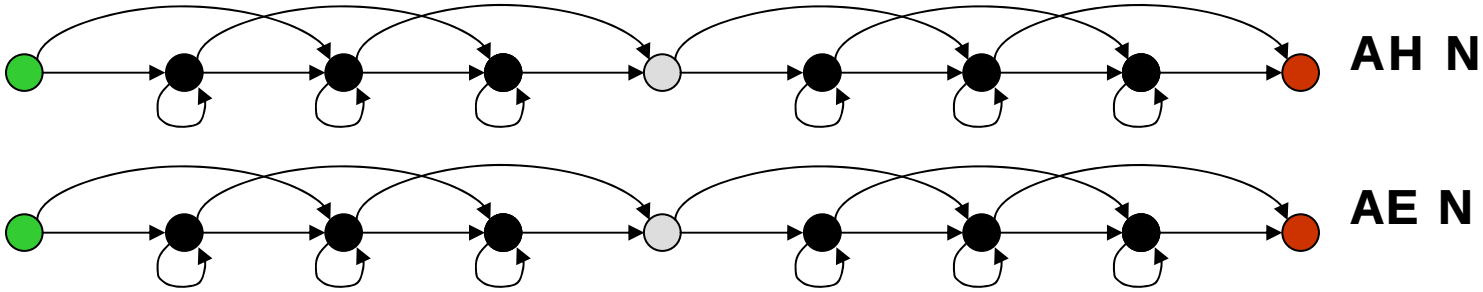
- Multiple pronunciations can be expressed compactly as a graph



- However, graph based representations can get very complex
  - often need introduction of non-emitting states

Multiple Pronunciations

- Typically, each of the pronunciations is simply represented by an independent HMM



- This implies, of course, that it is best to keep the number of alternate pronunciations of a word to be small
  - Do not include very rare pronunciations; they only confuse

Training Phoneme Models with SphinxTrain

- A simple exercise:
  - Train phoneme models using a small corpus
  - Recognize a small test set using these models