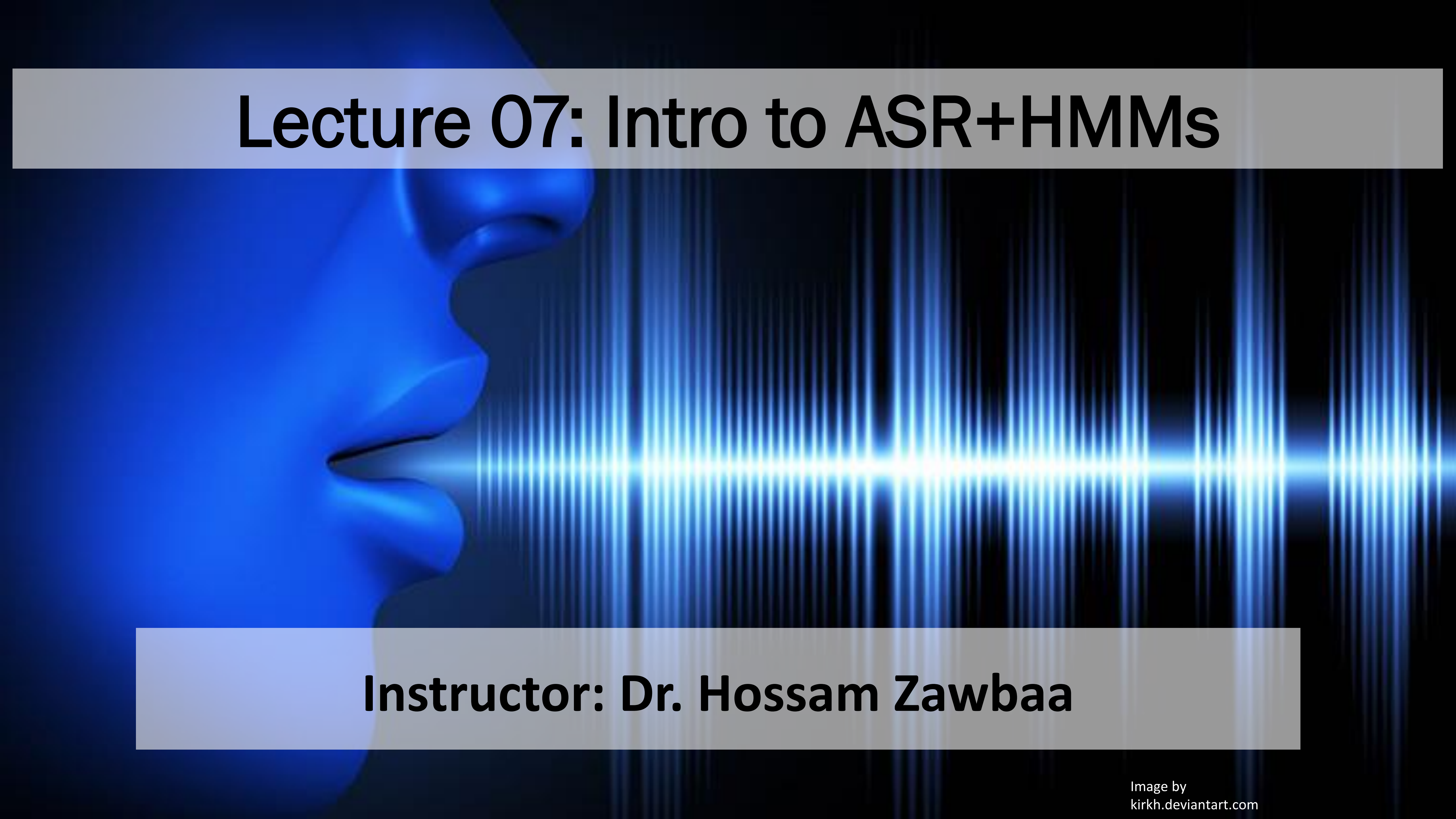
# Lecture 07: Intro to ASR+HMMs



Instructor: Dr. Hossam Zawbaa

Image by
kirkh.deviantart.com

# Current error rates

Exact numbers depend very much on the specific corpus

| Task | Vocabulary | Error Rate% |
|---|---|---|
| Digits | 11 | 0.5 |
| WSJ read speech | 5K | 3 |
| WSJ read speech | 20K | 3 |
| Broadcast news | 64,000+ | 10 |
| Conversational Telephone | 64,000+ | 20 |

# HSR versus ASR

**Human Speech Recognition (HSR) versus Automatic Speech Recognition (ASR)**

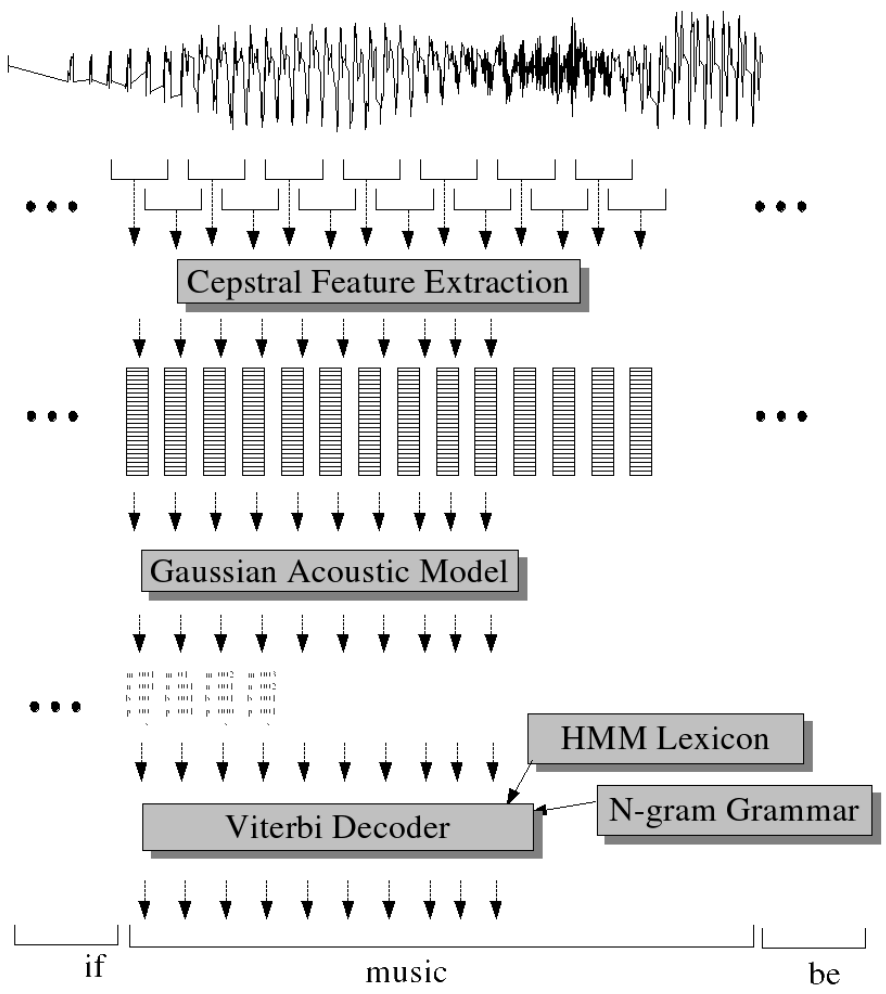
| Task | Vocab | ASR | HSR |
|---|---|---|---|
| Continuous digits | 11 | 0.5 | 0.009 |
| WSJ 1995 clean | 5K | 3 | 0.9 |
| WSJ 1995 w/noise | 5K | 9 | 1.1 |
| SWBD 2004 | 65K | 20 | 4 |

- Conclusions:
  - Machines about 5 times worse than humans
  - Gap increases with noisy speech
  - These numbers are rough

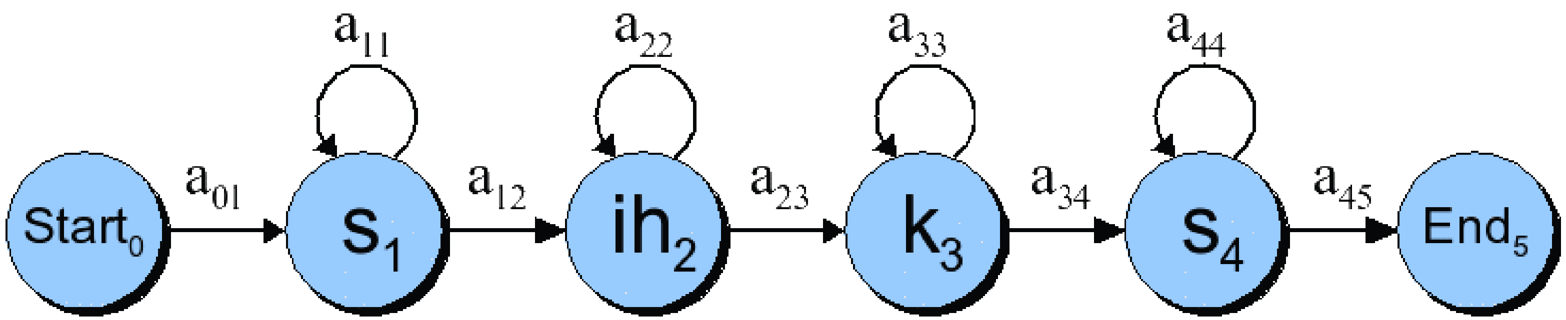
# ASR Design

- Build a statistical model of the speech-to-words process
- Collect lots and lots of speech, and transcribe all the words.
- Train the model on the labeled speech
- Paradigm: Supervised Machine Learning + Search

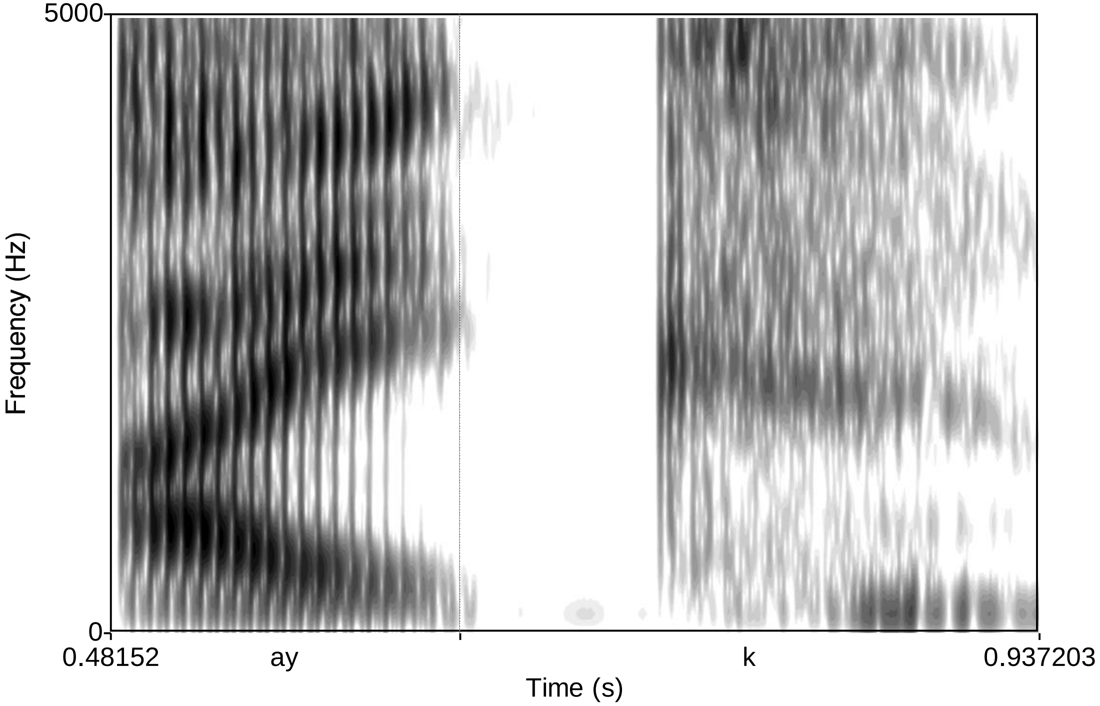
# Speech Recognition Architecture



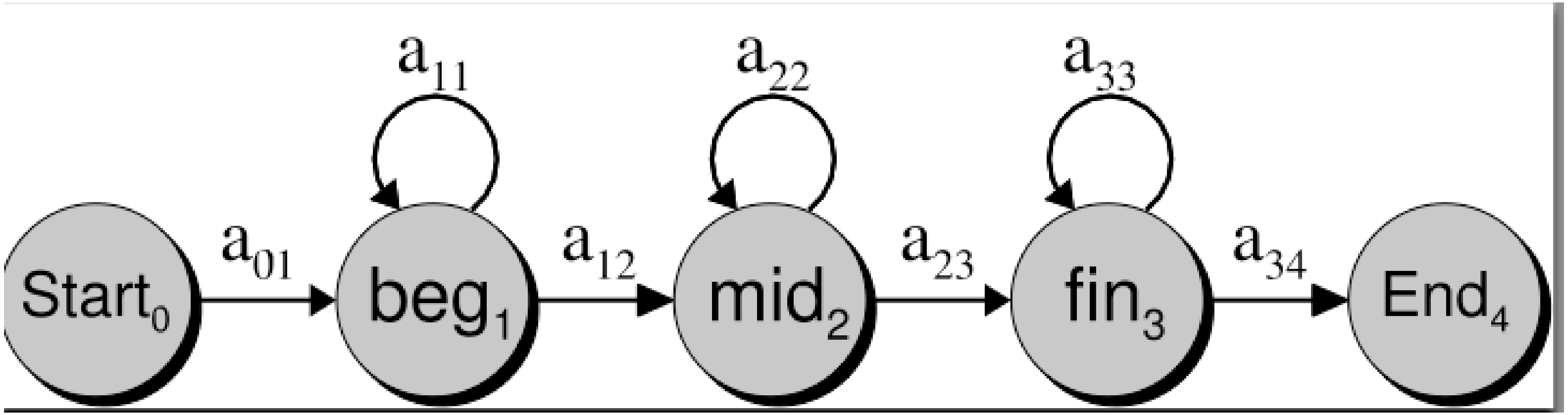
# HMMs for speech



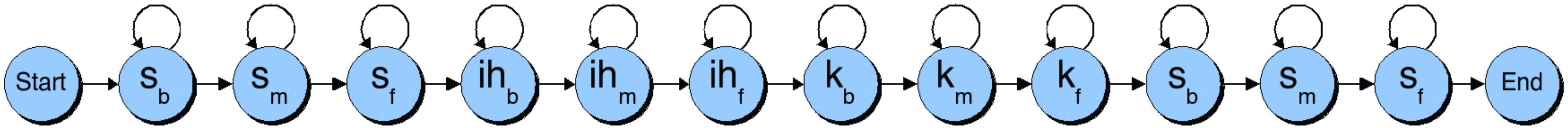
# Phones are not homogeneous!



# Each phone (voice/speech) has 3 sub-phones



# Resulting HMM word model for “six”



# HMMs more formally

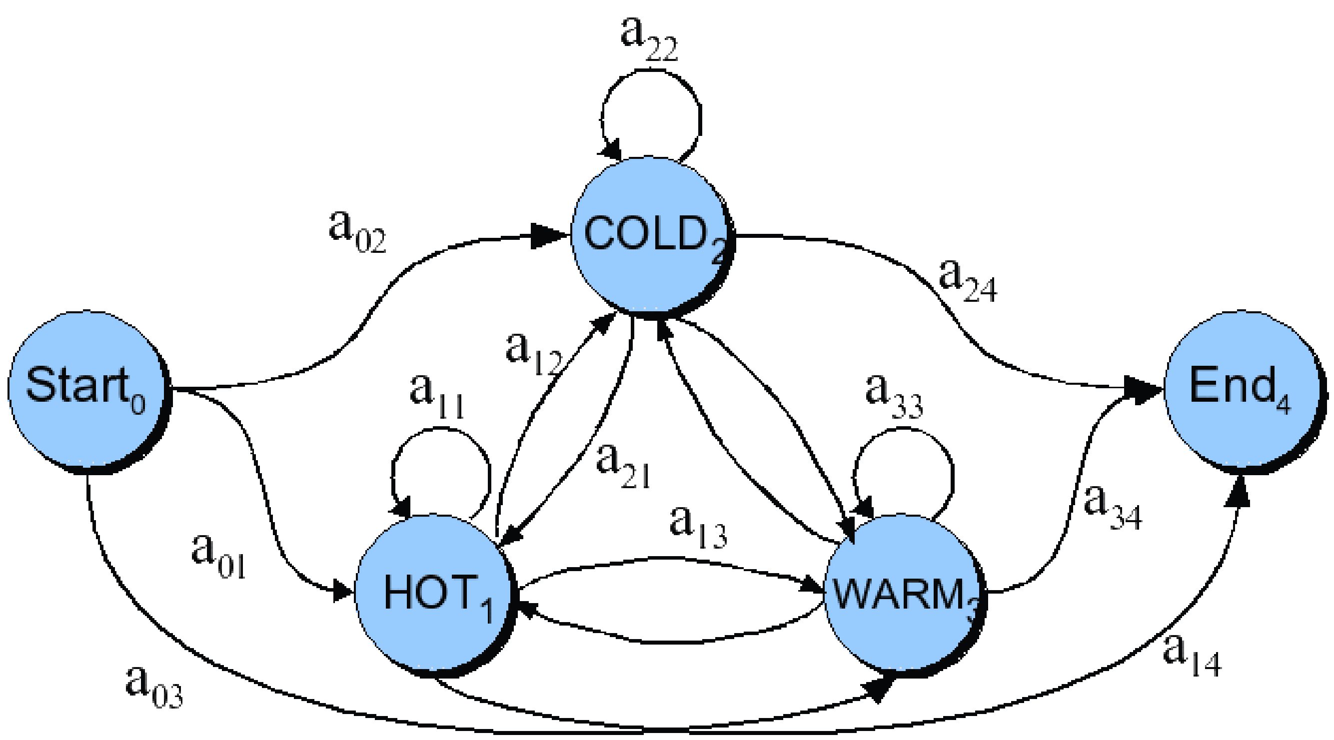
- Markov chains
- A kind of weighted finite-state automaton

| | |
|---|---|
| $Q = q_1 q_2 \ldots q_N$ | a set of **states** |
| $A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$ |
| $q_0, q_{end}$ | a special **start and end state** which are not associated with observations. |

**Markov Assumption:** $P(q_i | q_1 ... q_{i-1}) = P(q_i | q_{i-1})$

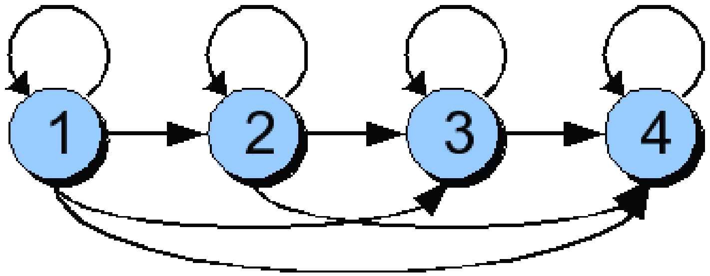
# HMMs more formally

- Markov chains
- A kind of weighted finite-state automaton

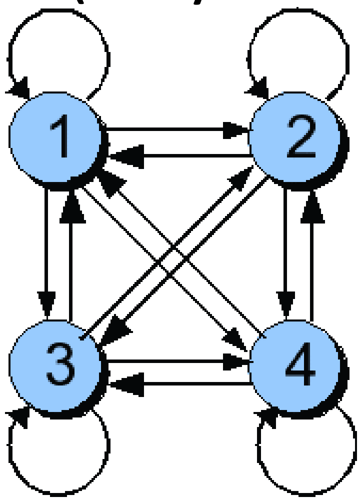


# Hidden Markov Models

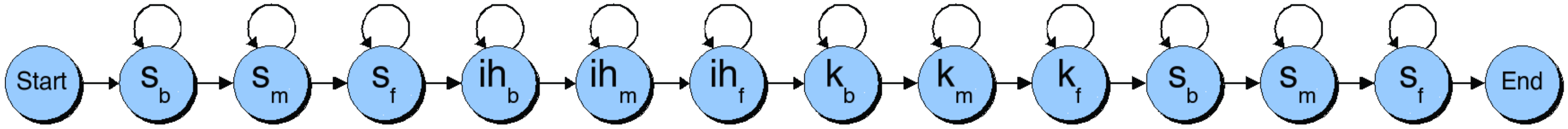
- Bakis network




Ergodic (fully-connected) network




- Left-to-right network



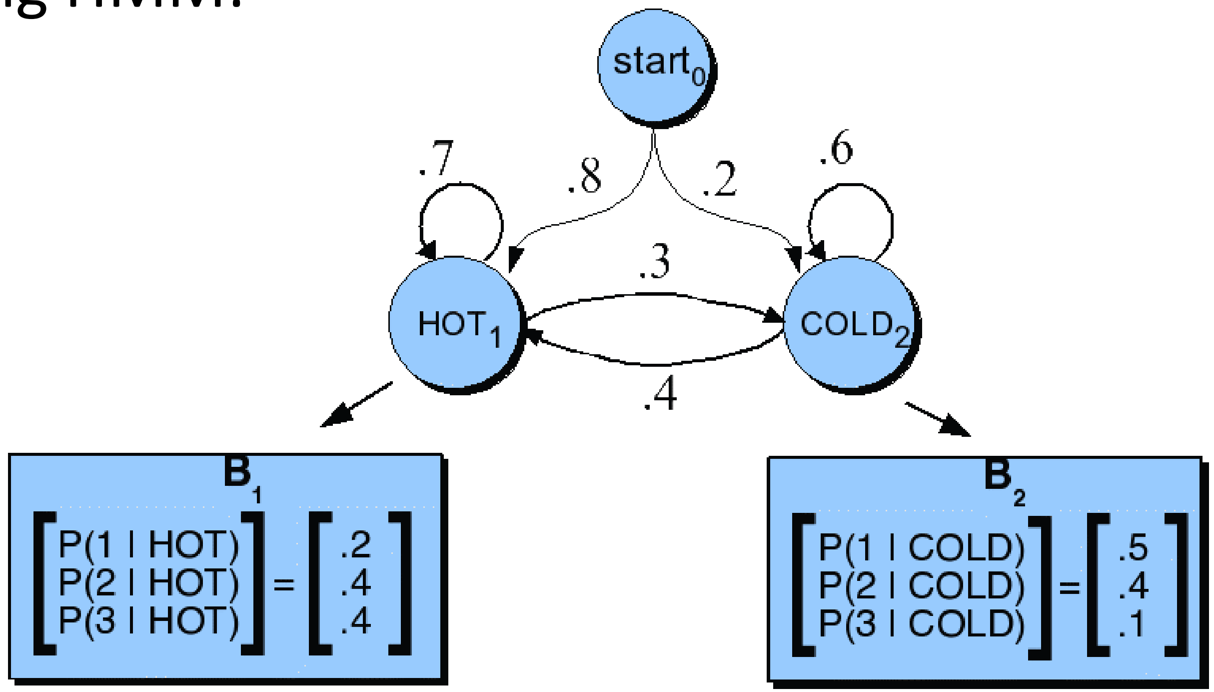
# The Three Basic Problems for HMMs

- Problem 1 (**Evaluation**)**:** Given the observation sequence $O=(o_1o_2 \ldots o_T)$, and an HMM model $\Phi = (A,B)$, how do we efficiently compute $P(O|\Phi)$, the probability of the observation sequence, given the model
- Problem 2 (**Decoding**)**:** Given the observation sequence $O=(o_1o_2 \ldots o_T)$, and an HMM model $\Phi = (A,B)$, how do we choose a corresponding state sequence $Q=(q_1q_2 \ldots q_T)$ that is optimal in some sense (i.e., best explains the observations)
- Problem 3 (**Learning**)**:** How do we adjust the model parameters $\Phi = (A,B)$ to maximize $P(O|\Phi)$?

# Problem 1: computing the observation likelihood

**Computing Likelihood:** Given an HMM $\lambda = (A, B)$ and an observation sequence $O$, determine the likelihood $P(O|\lambda)$.

- Given the following HMM:




- How likely is the sequence 3 1 3?

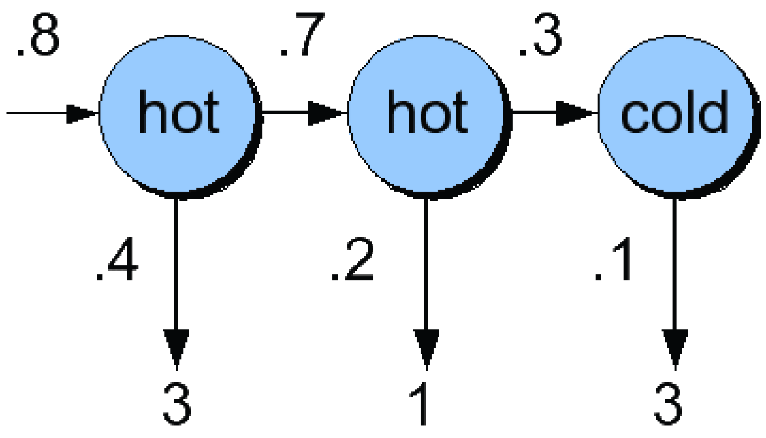
# How to compute likelihood

- For a Markov chain, we just follow the states 3 1 3 and multiply the probabilities
- But for an HMM, we don’t know what the states are!
- So let’s start with a simpler situation.
- Computing the observation likelihood for a **given** hidden state sequence
  - Suppose we knew the weather and wanted to predict how much ice cream Jason would eat.
  - I.e. P( 3 1 3 | H H C)

# Computing likelihood for 1 given hidden state sequence

$$P(O|Q) = \prod_{i=1}^{n} P(o_i|q_i) \times \prod_{i=1}^{n} P(q_i|q_{i-1})$$

$$\begin{aligned} P(3\ 1\ 3|\text{hot hot cold}) &= P(\text{hot}|\text{start}) \times P(\text{hot}|\text{hot}) \times P(\text{cold}|\text{hot}) \\ &\quad \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \end{aligned}$$



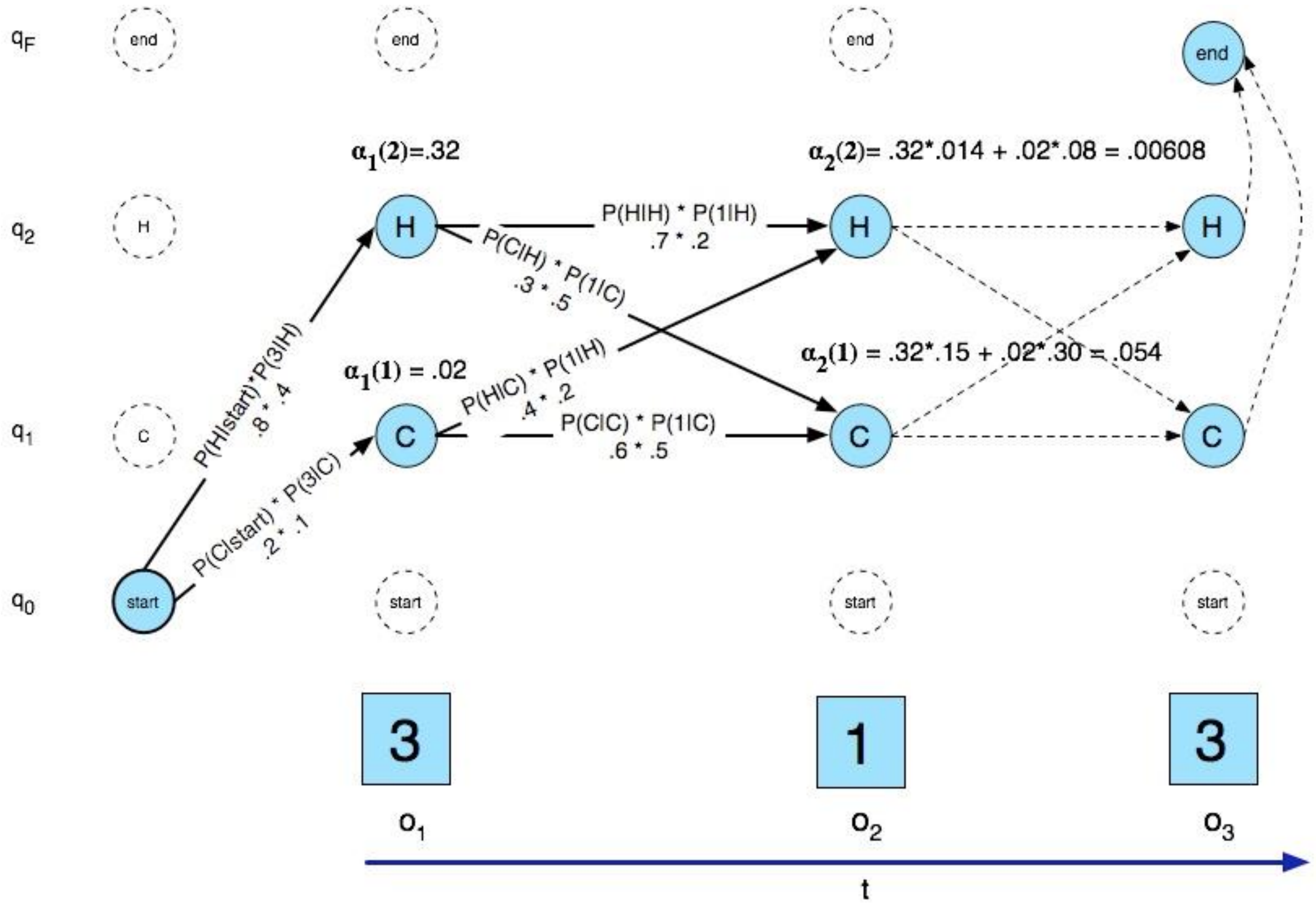
# Computing total likelihood of 3 1 3

- We would need to sum over
  - Hot hot cold
  - Hot hot hot
  - Hot cold hot
  - ….
- How many possible hidden state sequences are there for this sequence?
- How about in general for an HMM with N hidden states and a sequence of T observations?
  - $N^T$
- So we can’t just do separate computation for each hidden state sequence.

# Instead: the Forward algorithm

- A kind of **dynamic programming** algorithm
  - Uses a table to store intermediate values
- Idea:
  - Compute the likelihood of the observation sequence
  - By summing over all possible hidden state sequences
  - But doing this efficiently
    - By folding all the sequences into a single **trellis**

# The Forward Trellis



# The forward algorithm

- Each cell of the forward algorithm trellis $\text{alpha}_t(j)$
  - Represents the probability of being in state j
  - After seeing the first t observations
  - Given the automaton
- Each cell t

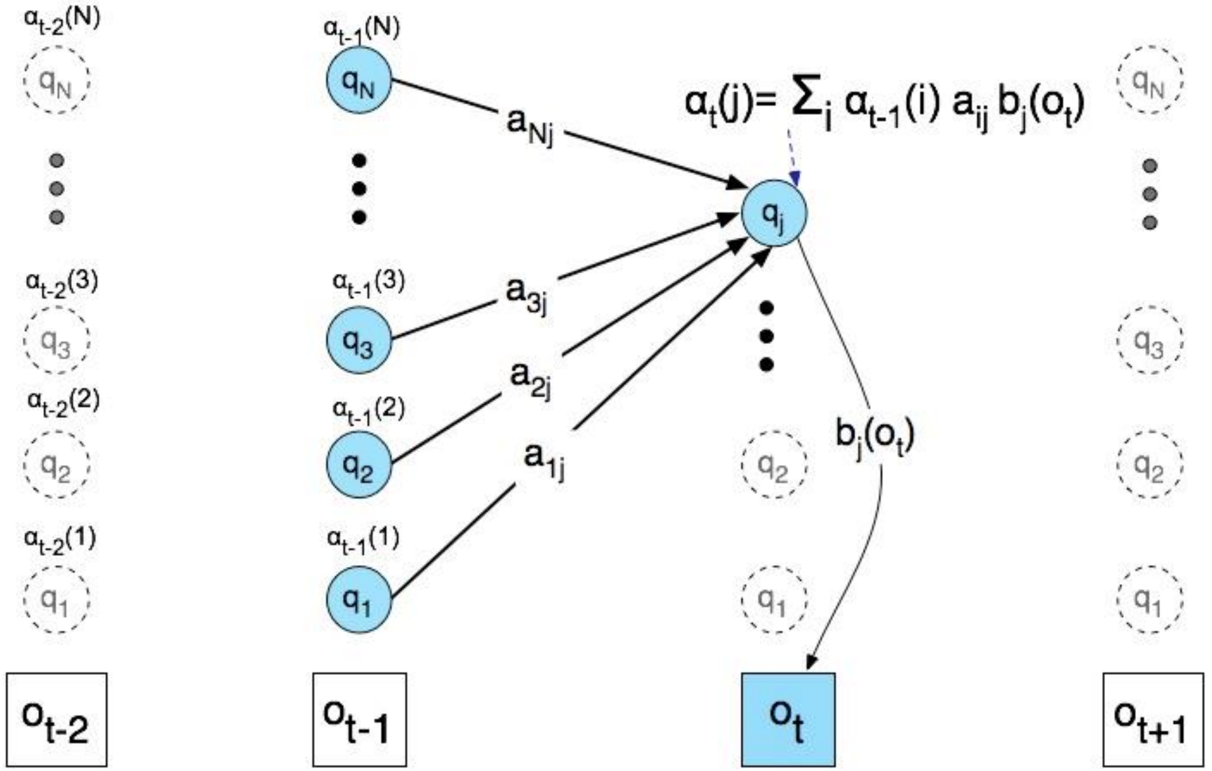
$$\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$$

# We update each cell

$\alpha_{t-1}(i)$ — the **previous forward path probability** from the previous time step

$a_{ij}$ — the **transition probability** from previous state $q_i$ to current state $q_j$

$b_j(o_t)$ — the **state observation likelihood** of the observation symbol $o_t$ given the current state $j$



# Decoding

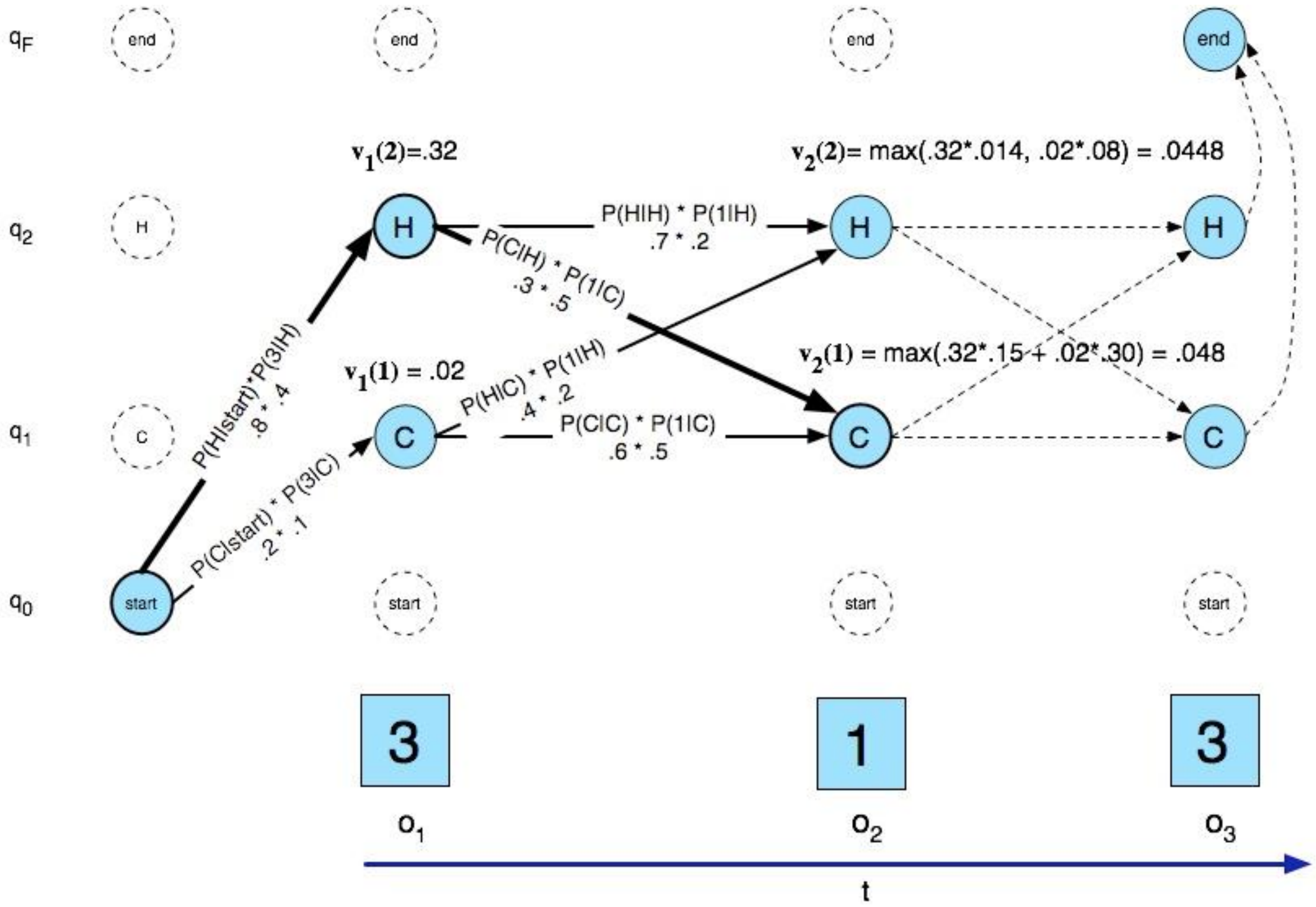
- Given an observation sequence
  - 3 1 3
- And an HMM
- The task of the **decoder**
  - To find the best **hidden** state sequence
- Given the observation sequence $O=(o_1o_2 \ldots o_T)$, and an HMM model $\Phi = (A,B)$, how do we choose a corresponding state sequence $Q=(q_1q_2 \ldots q_T)$ that is optimal in some sense (i.e., best explains the observations)

# Decoding

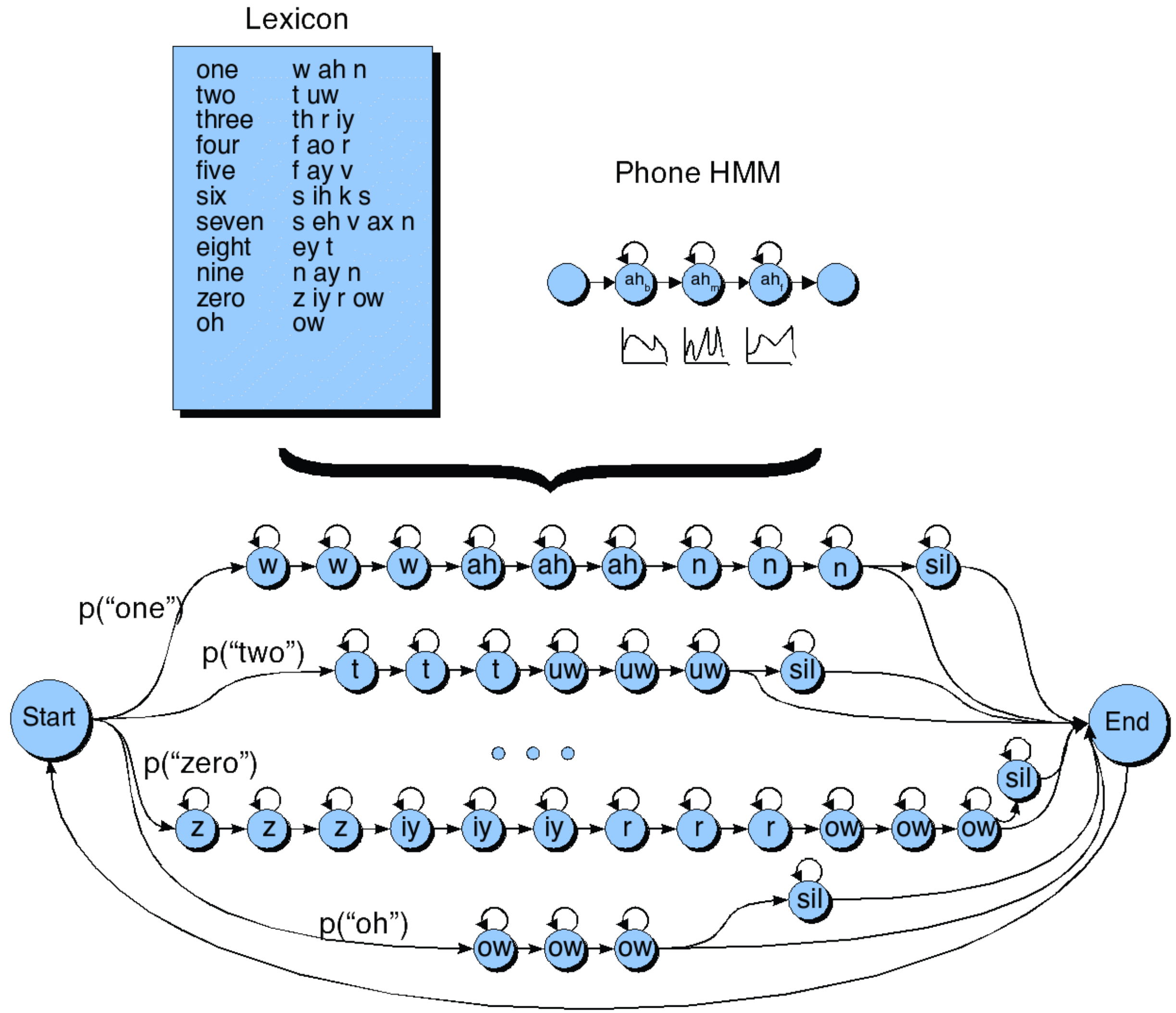
- One possibility:
  - For each hidden state sequence
    - HHH, HHC, HCH,
  - Run the forward algorithm to compute P($\Phi$ |O)
- Why not?
  - $N^T$
- Instead:
  - The Viterbi algorithm
  - Is again a **dynamic programming** algorithm
  - Uses a similar trellis to the Forward algorithm

# The Viterbi trellis

- The **Viterbi algorithm** is a dynamic programming algorithm for finding the most likely sequence of hidden states—called the **Viterbi path**—that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models.



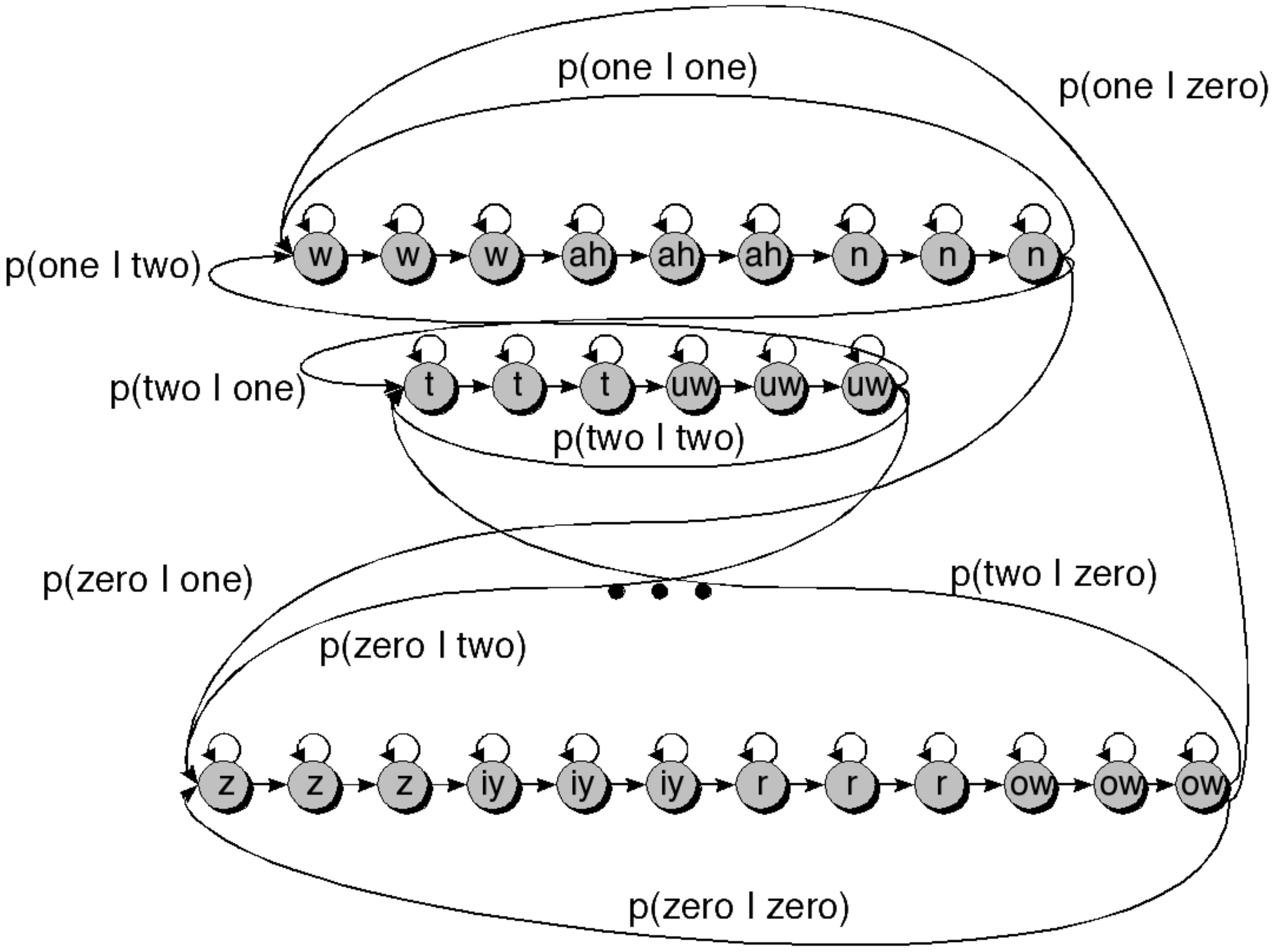
# HMM for digit recognition task

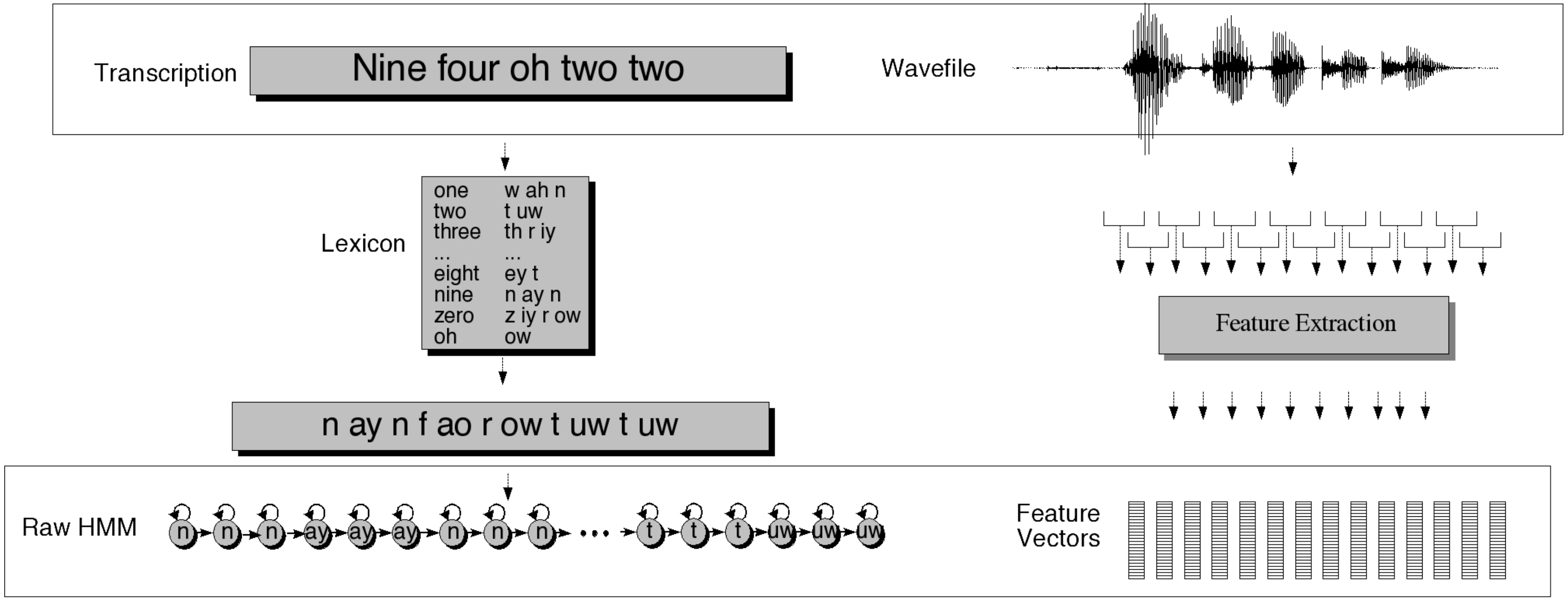


# The Evaluation (forward) problem for speech

- The observation sequence O is a series of Mel Frequency Cepstral Coefficient (MFCC) vectors
- The hidden states W are the phones and words
- For a given phone/word string W, our job is to evaluate P(O|W)

# Search space with bigrams




Transcription
Nine four oh two two
Wavefile
Lexicon
one w ah n
two t uw
three th r iy
... ...
eight ey t
nine n ay n
zero z iy r ow
oh ow
n ay n f ao r ow t uw t uw
Feature Extraction
Raw HMM
n n n ay ay ay n n n ... t t t uw uw uw
Feature Vectors

# Evaluation

- How to evaluate the word string output by a speech recognizer?

# Word Error Rate

- **Word Error Rate = 100 (Insertions + Substitutions + Deletions) / Total Word in Correct Transcript**

Alignment example:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| REF: | portable | **** | PHONE | UPSTAIRS | last | night | so |
| HYP: | portable | FORM | OF | STORES | last | night | so |
| Eval: | | I | S | S | | | |

**WER = 100 (1+2+0)/6 = 50%**

# NIST sctk-1.3 scoring softare: Computing WER with sclite

- http://www.nist.gov/speech/tools/
- Sclite aligns a hypothesized text (HYP) (from the recognizer) with a correct or reference text (REF) (human transcribed)

```
id: (2347-b-013)
Scores: (#C #S #D #I) 9 3 1 2
REF:  was an engineer SO I   i was always with **** **** MEN UM   and they
HYP:  was an engineer ** AND i was always with THEM THEY ALL THAT and they
Eval:                 D  S                     I    I    S   S
```

**WER = 100 (2+3+1)/13 = 46.15%**