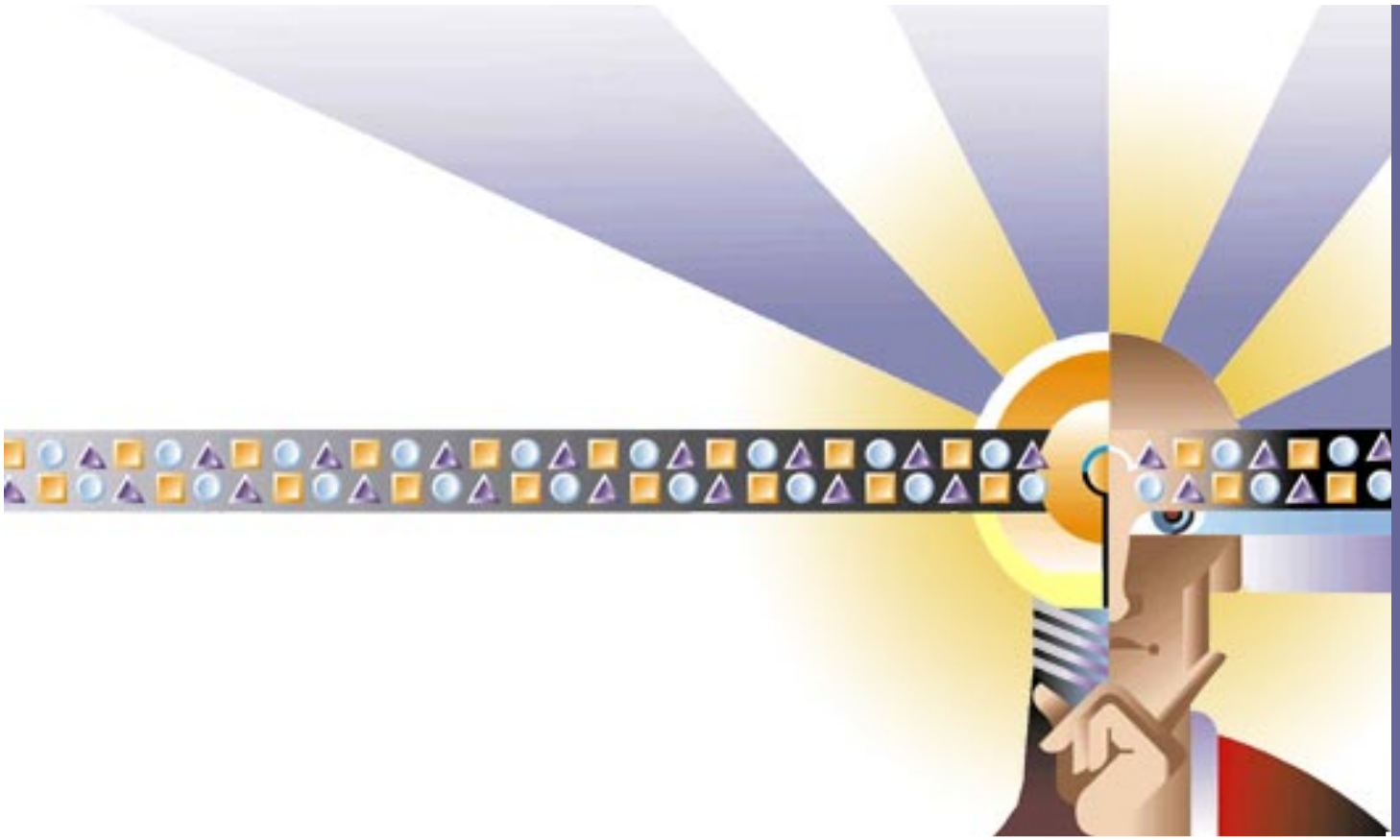


Sphinx 4

MIT Lunch Discussion

December 18, 2002

Sphinx 4 Team

Project at a Glance – Sphinx 4

Description

- Speech recognition written entirely in the Java™ programming language
- Based upon Sphinx developed at CMU:

  www.speech.cs.cmu.edu/sphinx/

Goals

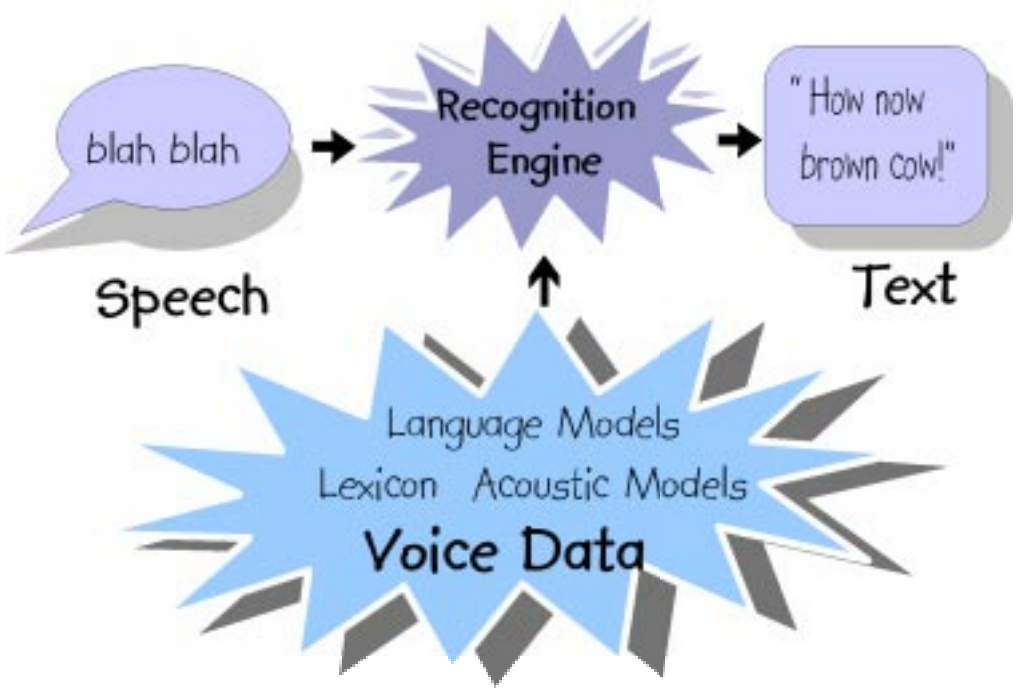
- Highly flexible recognizer
- Performance equal to or exceeding Sphinx 3
- Collaborate with researchers at CMU and MERL, Sun Microsystems and others

Project at a Glance – Sphinx 4

Distinctives:

- Highly configurable front-end processing
- Support isolated word, n-gram and context free grammars
- Support for arbitrary unit context sizes to allow for improved recognition
- Pluggable architecture allows new search and pruning algorithms to be used

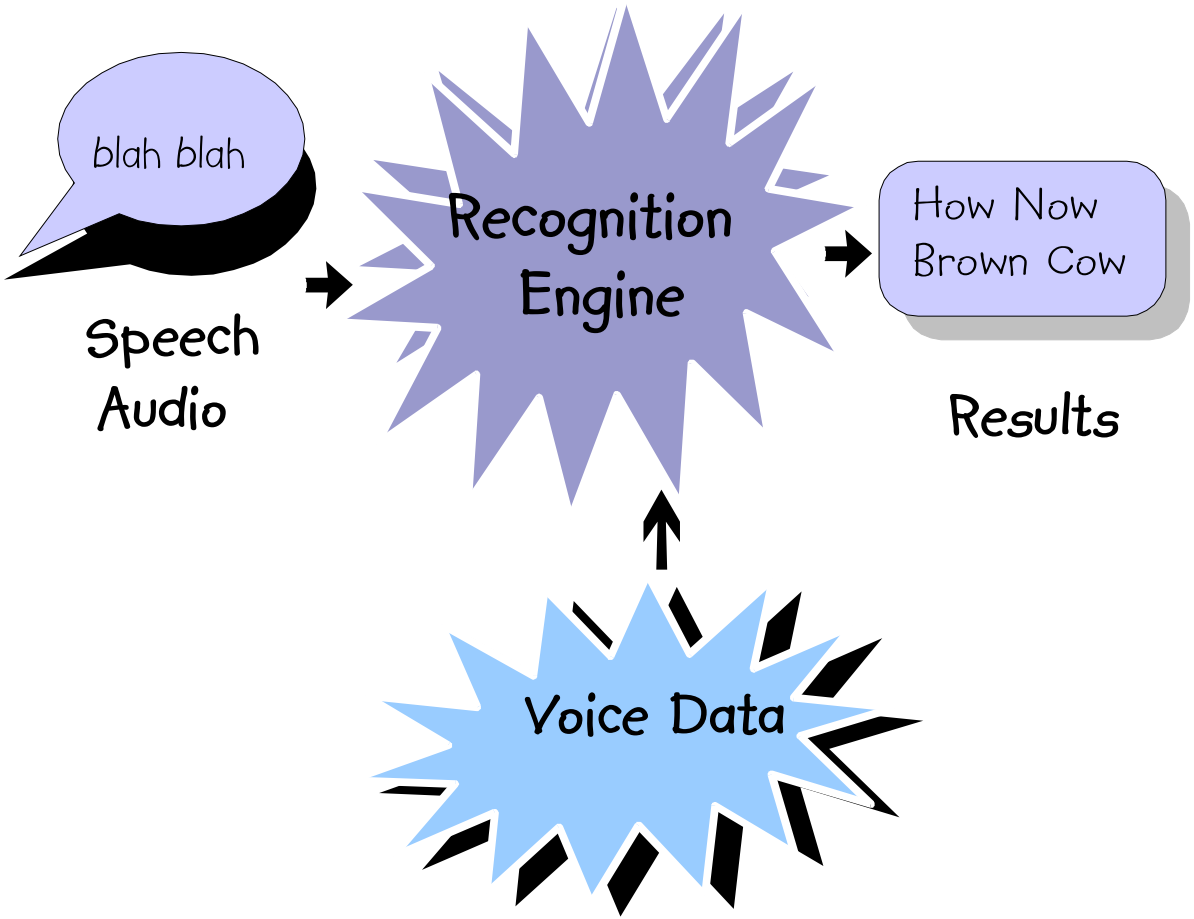
Recognition Issues



- Quality of recognition is directly related to quality of voice data
- As part of the Sphinx 4 project we will be developing a trainer to give us good voice data

Good Voice Data is the key to good recognition!

How does a Recognizer Work?



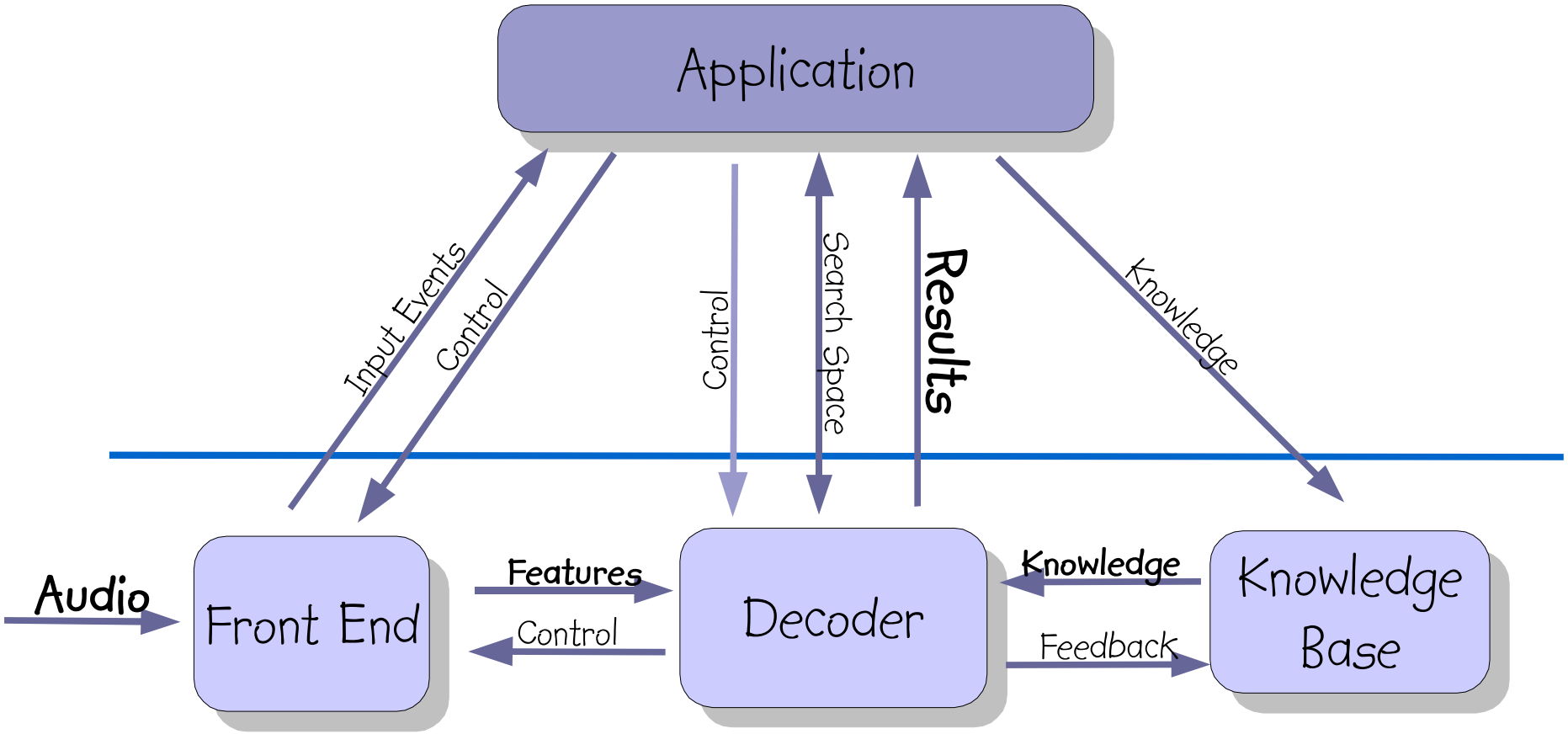
Goal:

- Audio goes in
- Results come out

Three application types

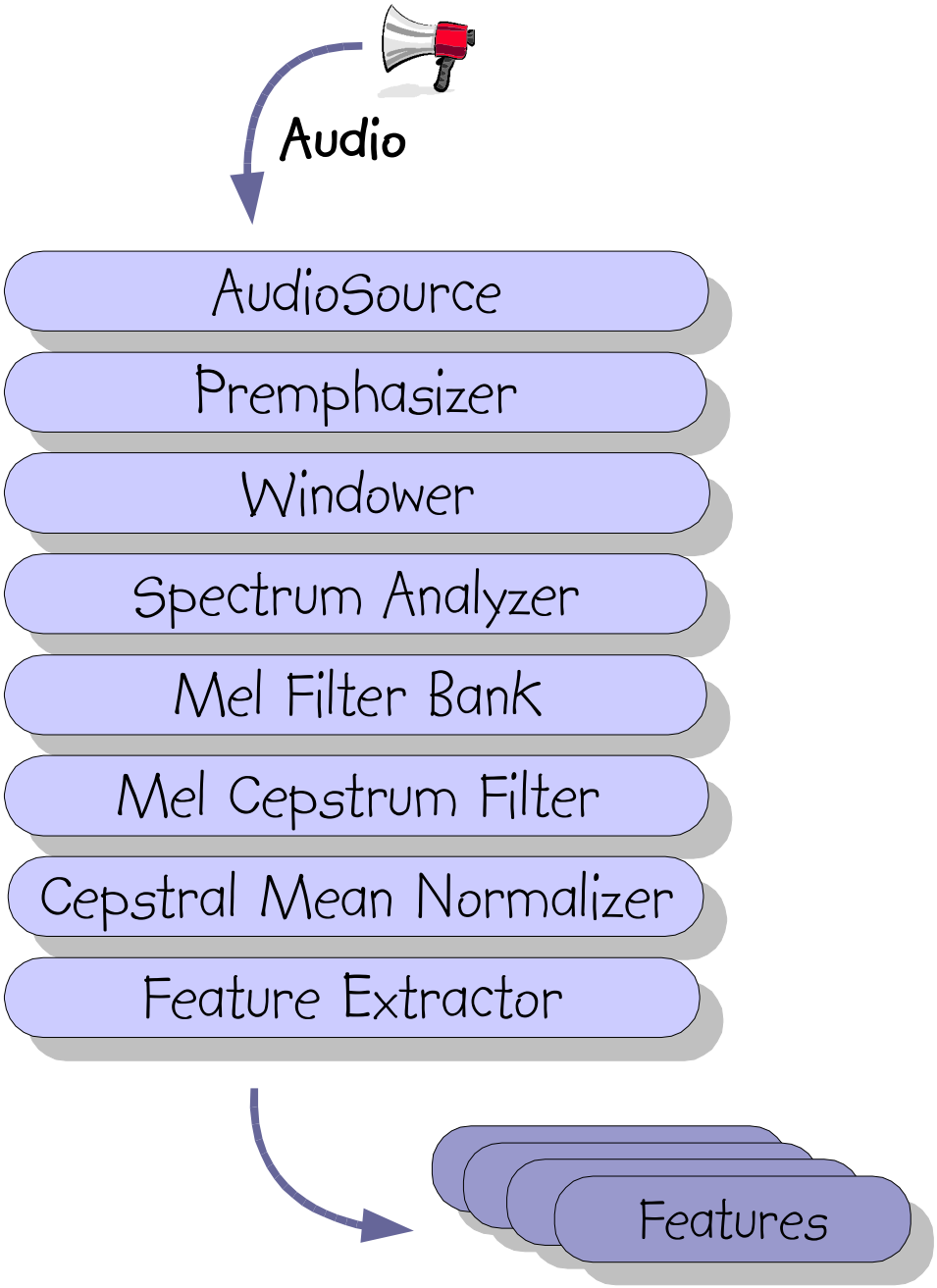
- Isolated words
- Command / Control
- General Dictation

Sphinx 4 Architecture



Front-End

- Transforms speech waveform into features used by recognition
- Features are sets of mel-frequency cepstrum coefficients (MFCC)
- MFCC model human auditory system
- Front-End is a set of signal processing filters
- Pluggable architecture

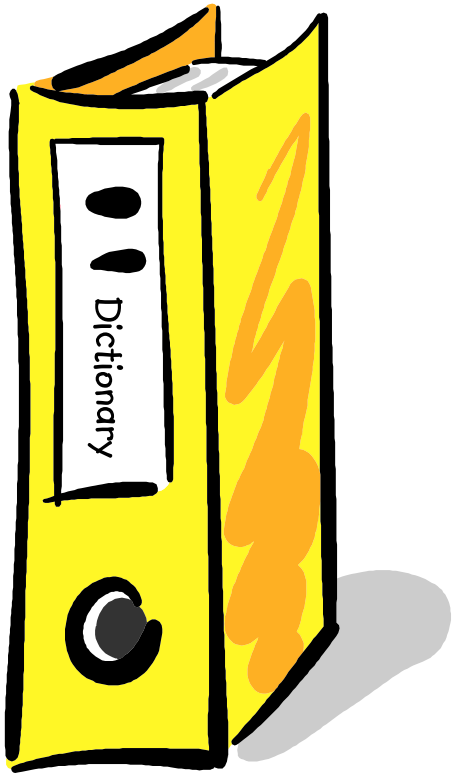


Knowledge Base

- The data that drives the decoder
- Consists of three sets of data:
  - Dictionary
  - Acoustic Model
  - Language Model
- Needs to scale between the three application types

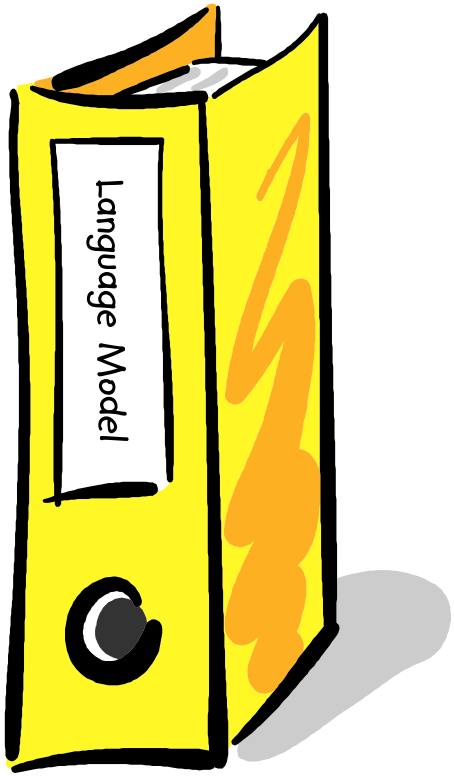
Dictionary

- Maps words to pronunciations
- Provides word classification information (such as part-of-speech)
- Single word may have multiple pronunciations
- Pronunciations represented as phones or other units
- Can vary in size from a dozen words to >100,000 words



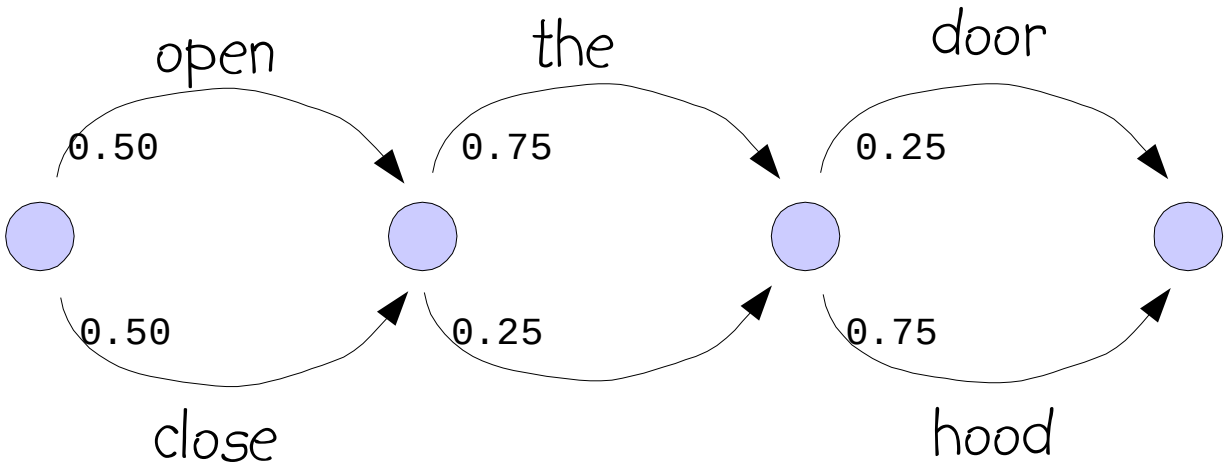
Language Model

- Describes what is likely to be spoken in a particular context
- Uses stochastic approach. Word transitions are defined in terms of transition probabilities
- Helps to constrain the search space



Command Grammars

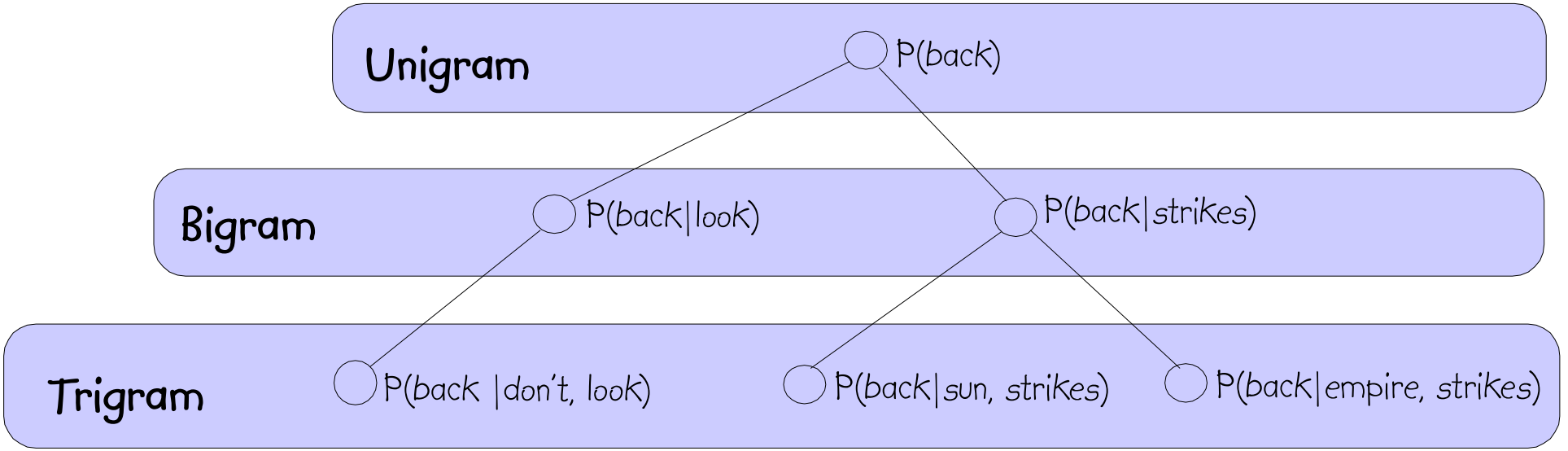
- Probabilistic context-free grammar
- Relatively small number of states
- Appropriate for command and control applications




```
open|close [the] door|hood
```

N-gram Language Models

- Probability of word N dependent on word N-1, N-2, ...
- Bigrams and trigrams most commonly used
- Used for large vocabulary applications such as dictation
- Typically trained by very large (millions of words) corpus

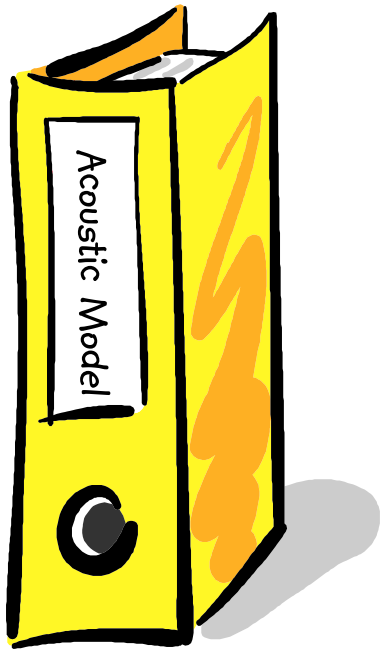


Language Model Issues

- Pruning
- Smoothing
- Adaptation
- Training (50 to 100 million words)
- Finite State Transducers

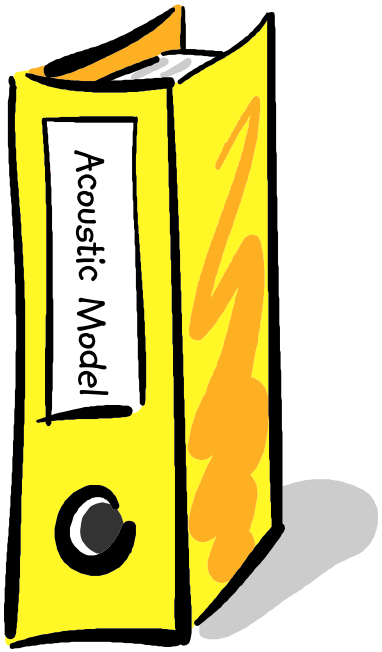
Acoustic Models

- Database of statistical models
- Each statistical model represents a single unit of speech such as a word or phoneme
- Acoustic Models are created/trained by analyzing large corpora of labeled speech
- Acoustic Models can be speaker dependent or speaker independent



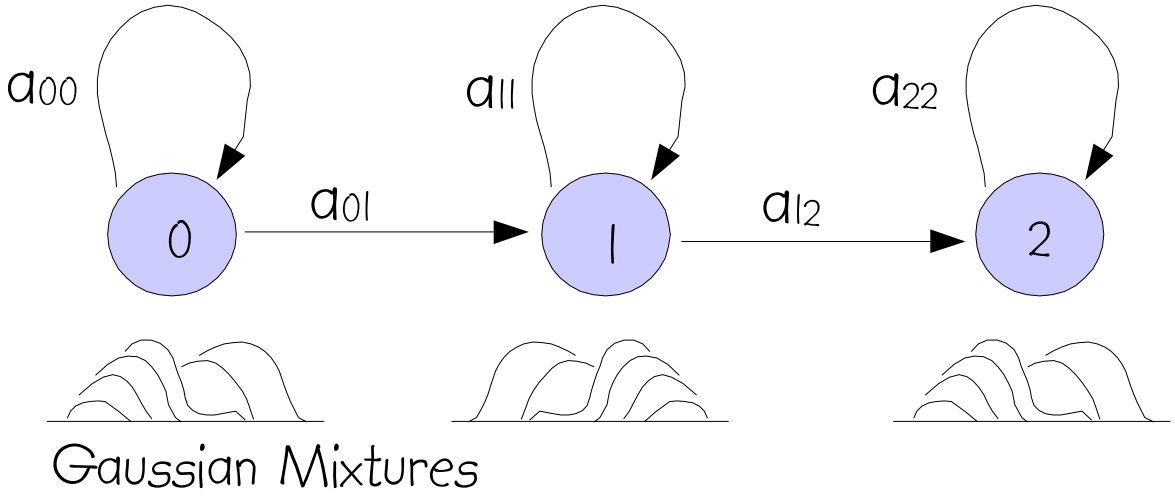
Hidden Markov Model

- Hidden Markov Models represent each unit of speech in the Acoustic Model
- HMMs are used by a scorer to calculate the acoustic probability for a particular unit of speech
- A Typical HMM uses 3 states to model a single context dependent phoneme
- Each state of an HMM is represented by a set of Gaussian mixture density functions

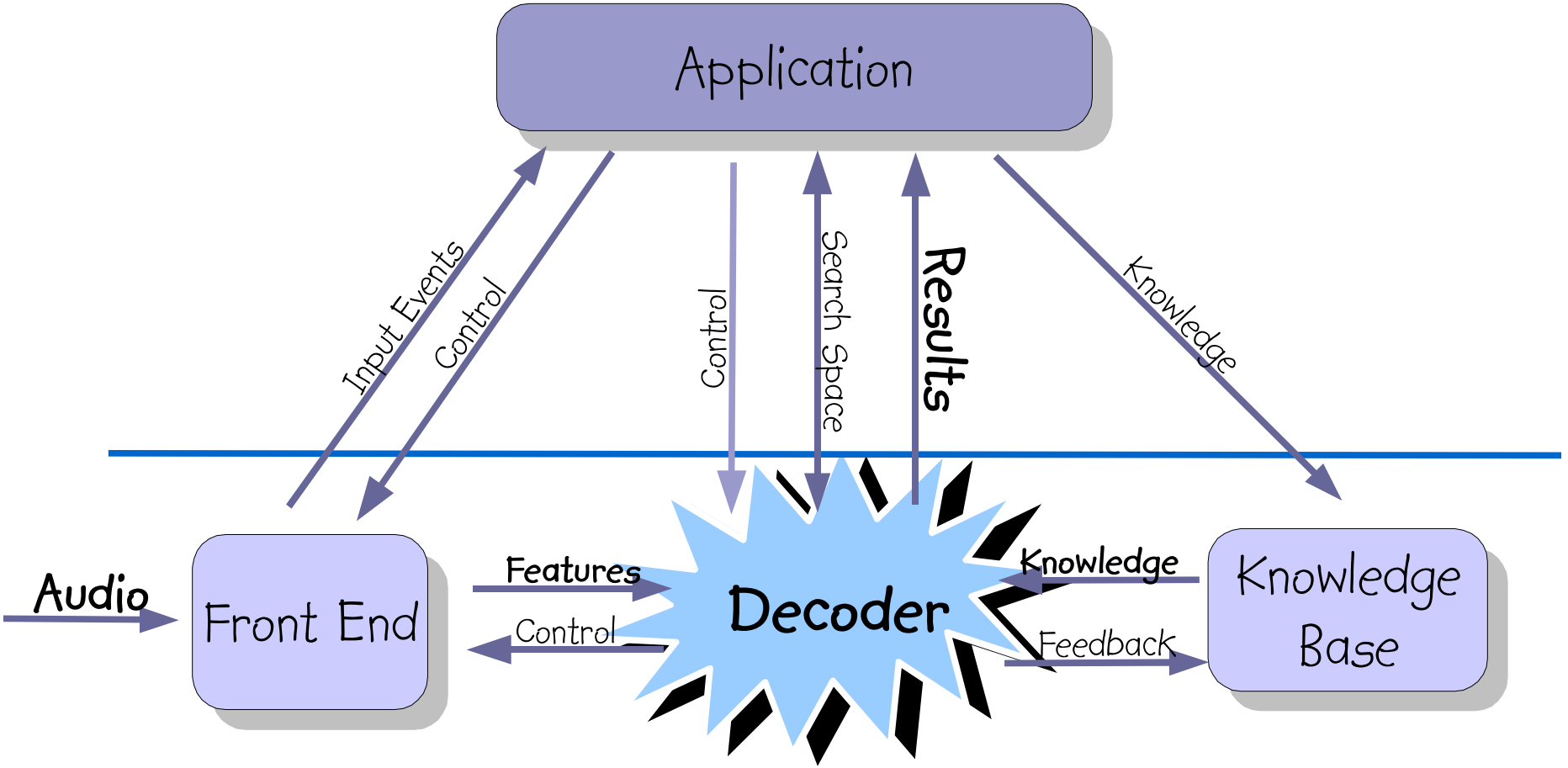


Gaussian Mixtures

- Gaussian mixture density functions represent each state in an HMM
- Each set of Gaussian mixtures is called a senone
- HMMs can share senones



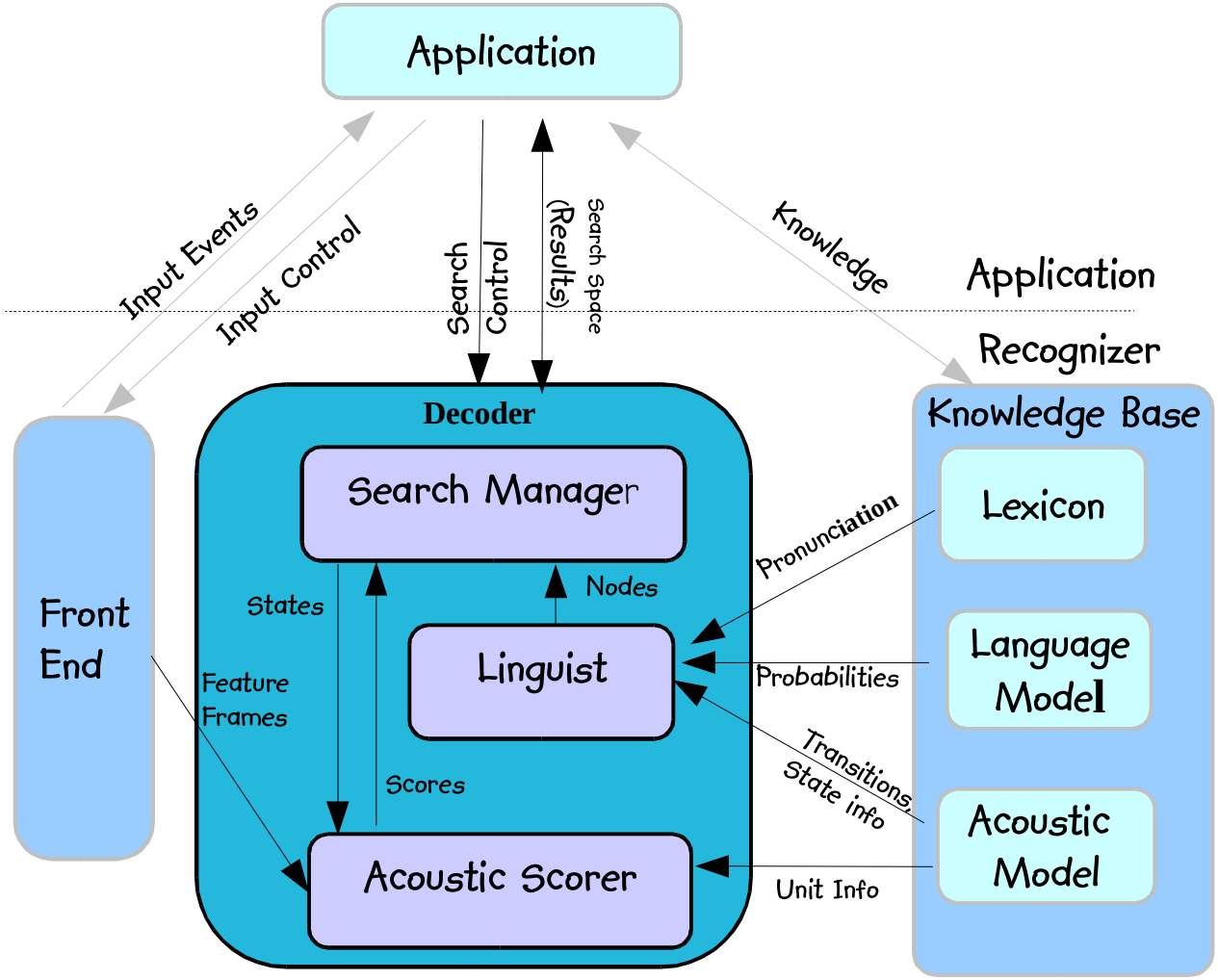
The Decoder


Application
Input Events
Control
Control
Search Space
Results
Knowledge
Audio
Front End
Features
Control
Decoder
Knowledge
Feedback
Knowledge Base

Decoder - heart of the recognizer

- Selects next set of likely states
- Scores incoming features against these states
- Prunes low scoring states
- Generates results

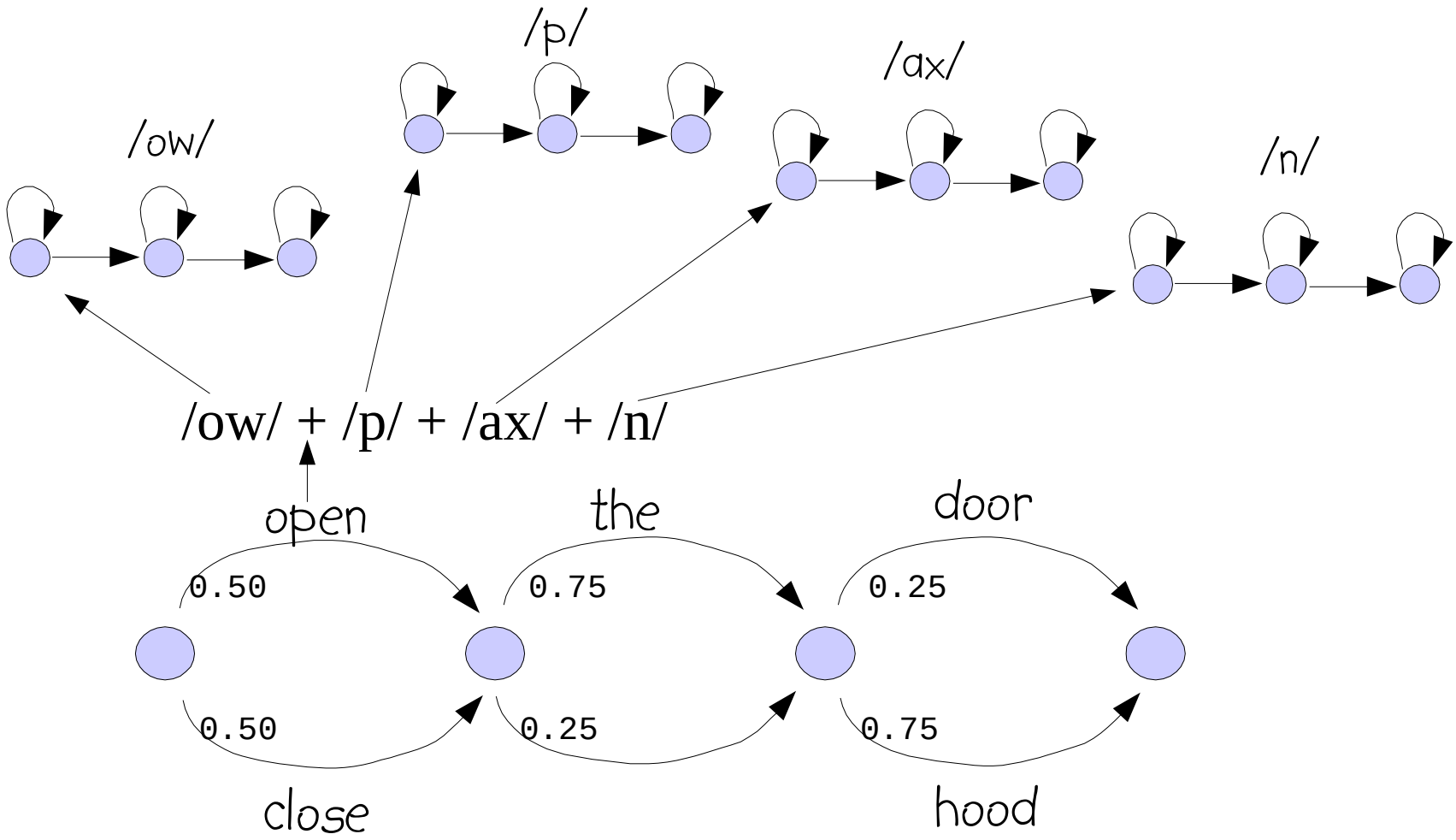
Decoder Overview



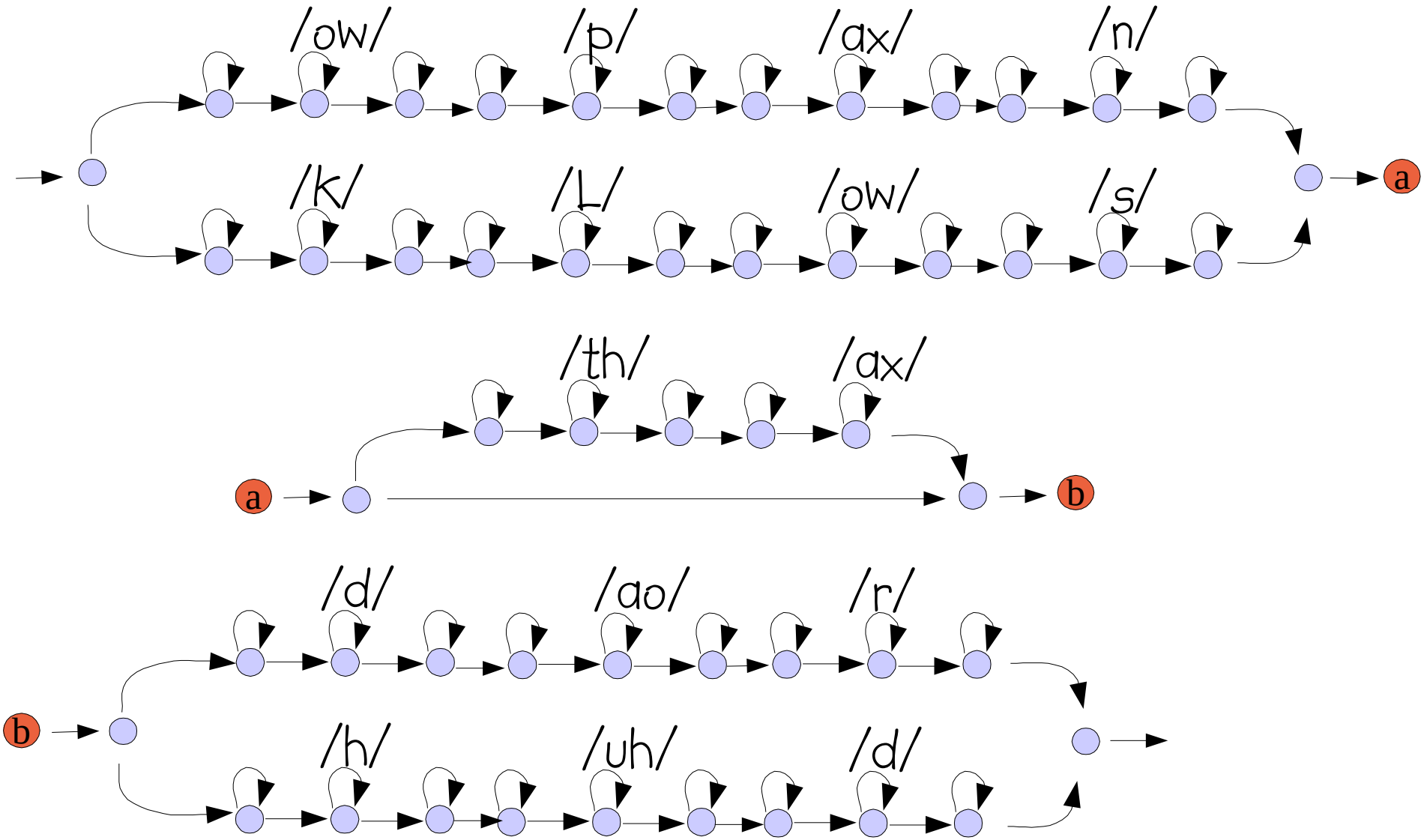
Selecting next set of states

- Uses Grammar to select next set of possible words
- Uses dictionary to collect pronunciations for words
- Uses Acoustic Model to collect HMMs for each pronunciation
- Uses transition probabilities in HMMs to select next set of states

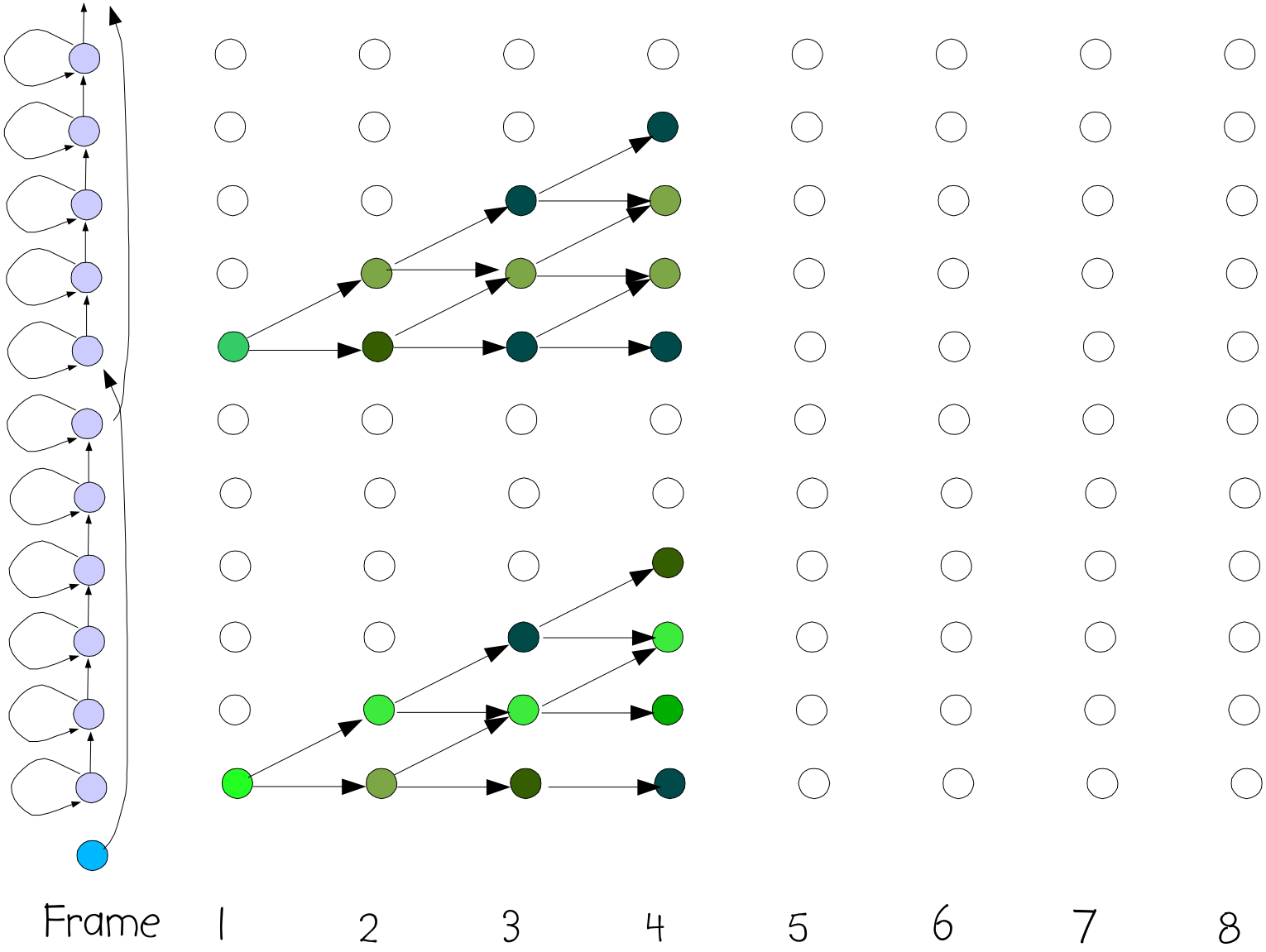
Sentence HMMS



Sentence HMMS



Search Lattice



Search Results

- Search results are just pointers back into the search lattice for high scoring branches
- Enough information is kept with every node in the lattice so unit, pronunciation, word, grammar and audio information can be reconstructed

Search Issues

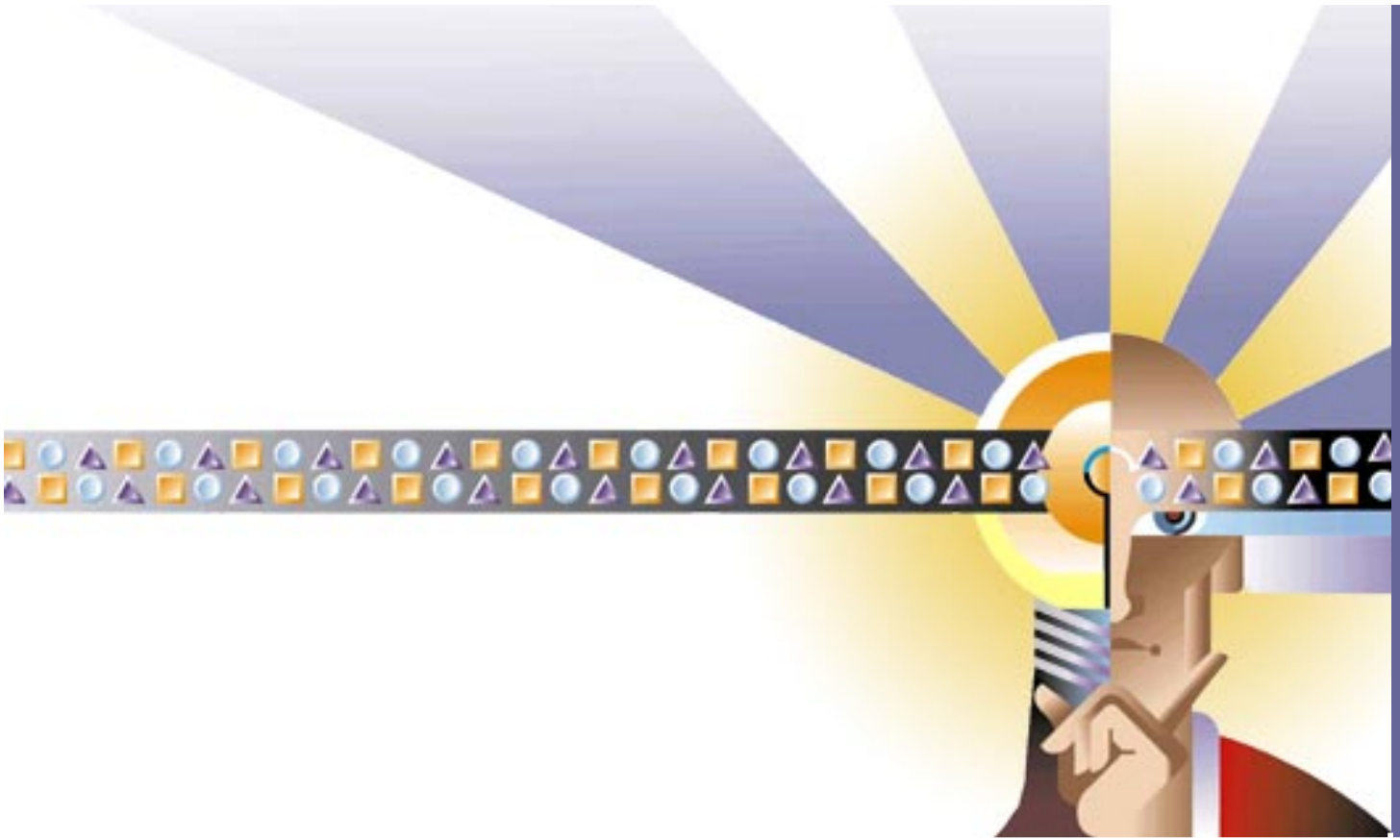
- Dealing with huge fan-out of search space
- Pruning strategies
- Reducing scoring calculations
  - Shared states
  - Sub-vector quantization

Java™ Platform Issues

- GC makes managing data much easier
- Native engines typically optimize inner loops for the CPU – can't do that on the Java platform
- Native engines arrange data to optimize cache hits – can't really do that either

Sphinx 4 Summary

- Speech recognition is a challenging problem for the Java platform
- Open source at SourceForge.net
- Accuracy currently matching S3 for most cases



Sphinx 4 Q&A