Sphinx-3 to 3.2

Mosur Ravishankar
School of Computer Science, CMU
Nov 19, 1999

Outline

- Recognition problem
  - Search for the most likely word sequence matching the input speech, given the various models
  - Illustrated using Sphinx-3 (original)
- Lextree search (Sphinx-3.2)
  - Search organization
  - Pruning
- Experiments
- Conclusion

Recognition Problem

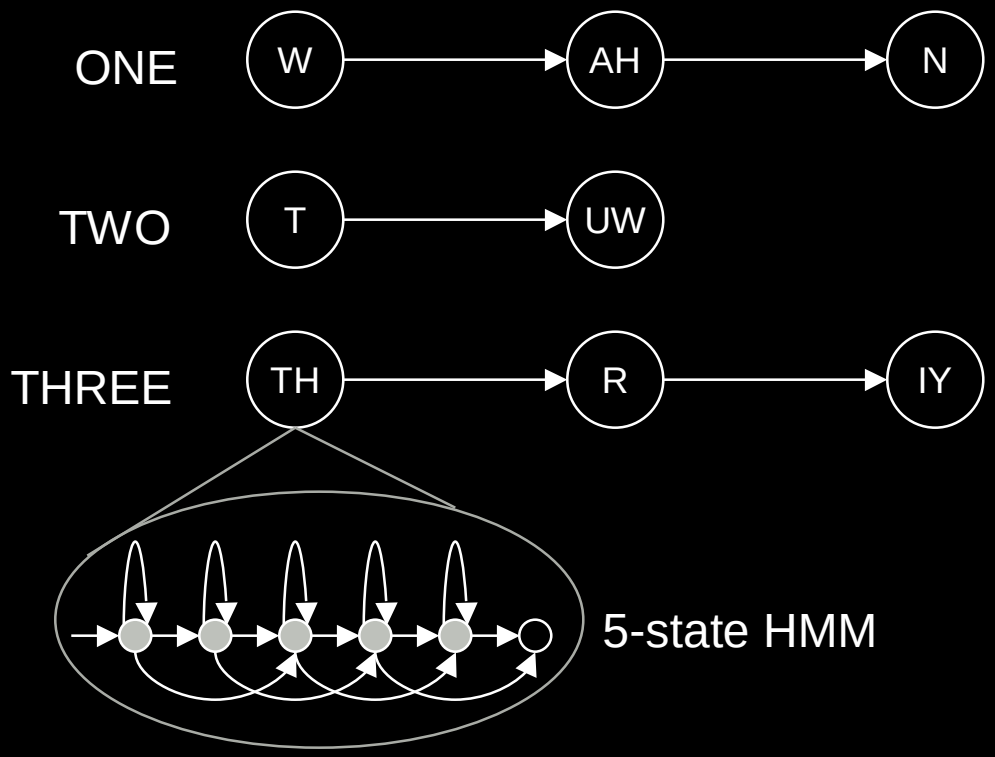
- Search organization
- Continuous speech recognition
- Cross-word triphone modeling
- Language model integration
- Pruning for efficiency

Search Organization in Sphinx-3

- Flat lexical structure
- Cross-word triphone modeling
  - Multiplexing at word beginning
  - Replication at word end
  - Single-phone words: combination of both
- LM score applied upon transition into word
  - Trigram language model
  - However, only single best history maintained
- Beam-based pruning
  - Long-tailed distribution of active HMMs/frame

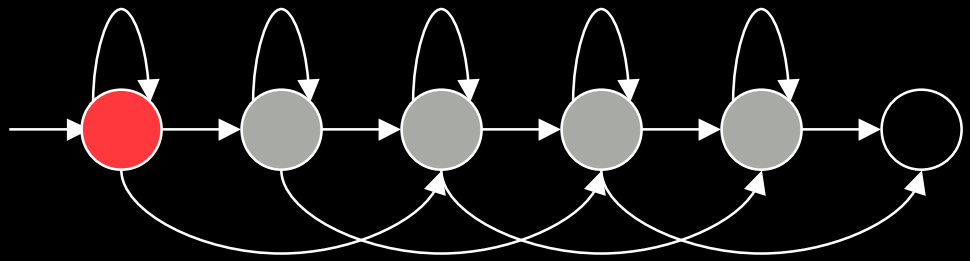
Sphinx-3 Lexical Structure

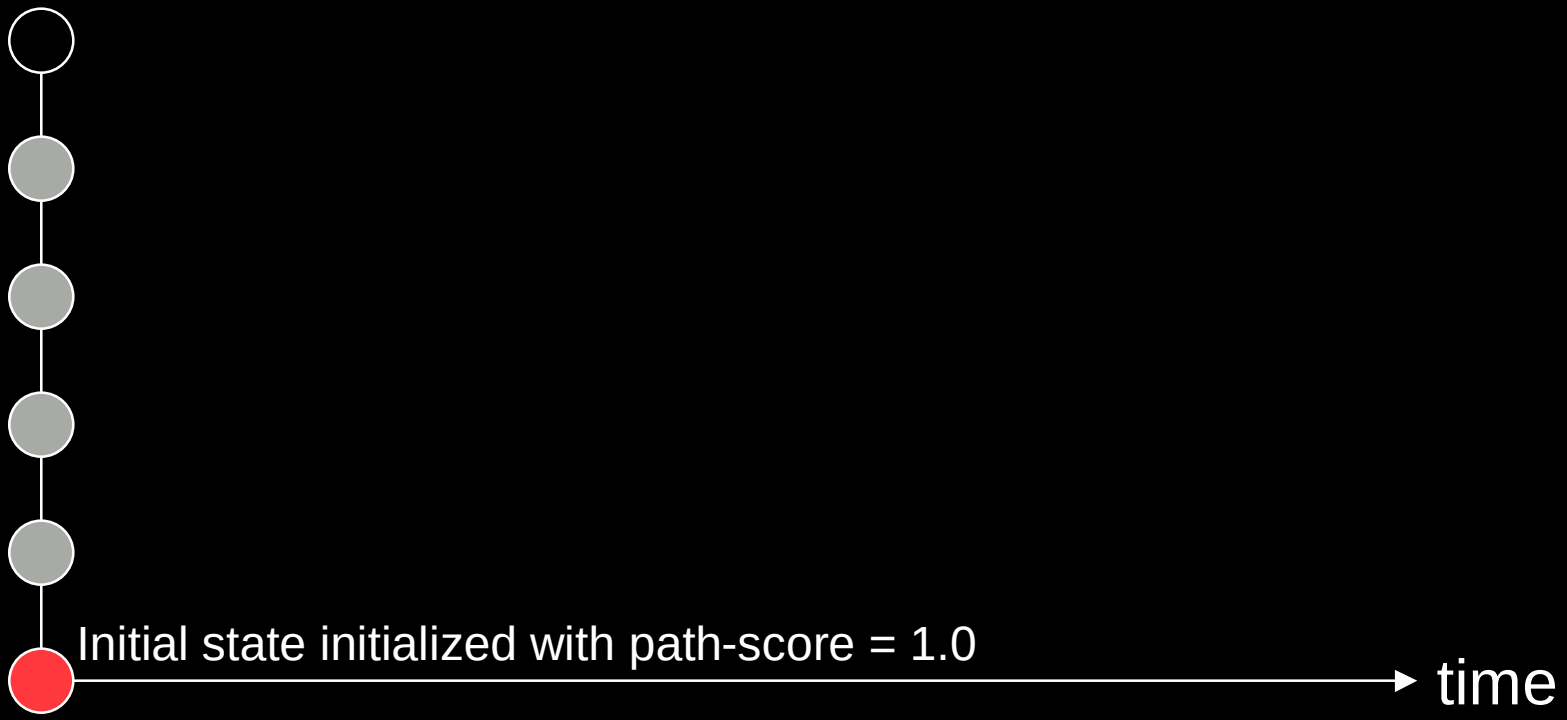
- **Flat lexicon; every word treated independently:**



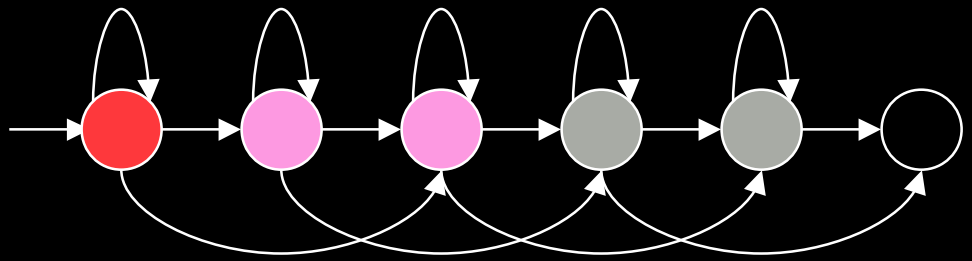
- **Evaluating an HMM w.r.t. input speech: Viterbi search**
  - **Score the best state-sequence through HMM, given the input**

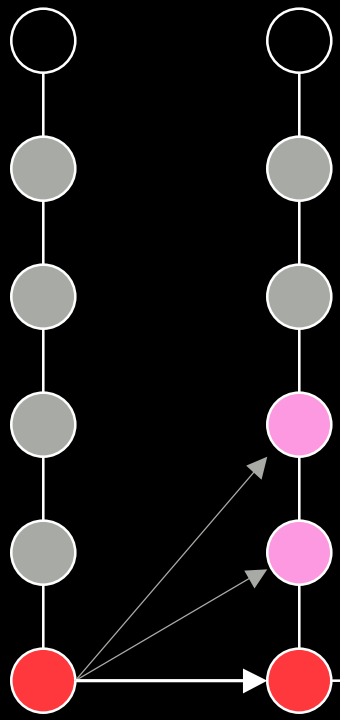
Viterbi Search





Viterbi Search (contd.)





State with best path-score
State with path-score < best
State without a valid path-score

$$P_j(t) = \max_i [P_i(t\text{-}1)\, a_{ij}\, b_j(t)]$$

State transition probability, i to j

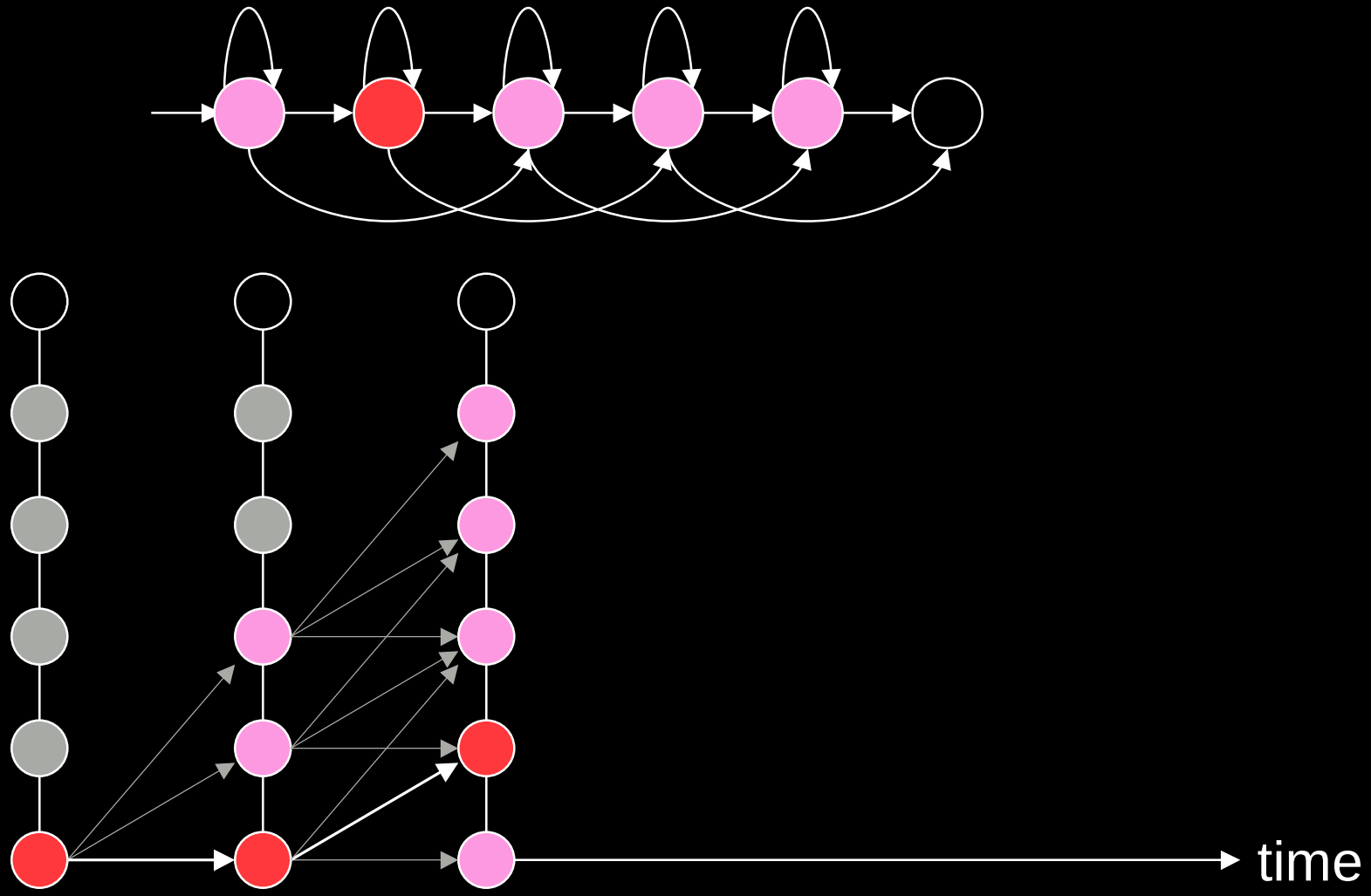
Score for state j, given the input at time t

Total path-score ending up at state j at time t

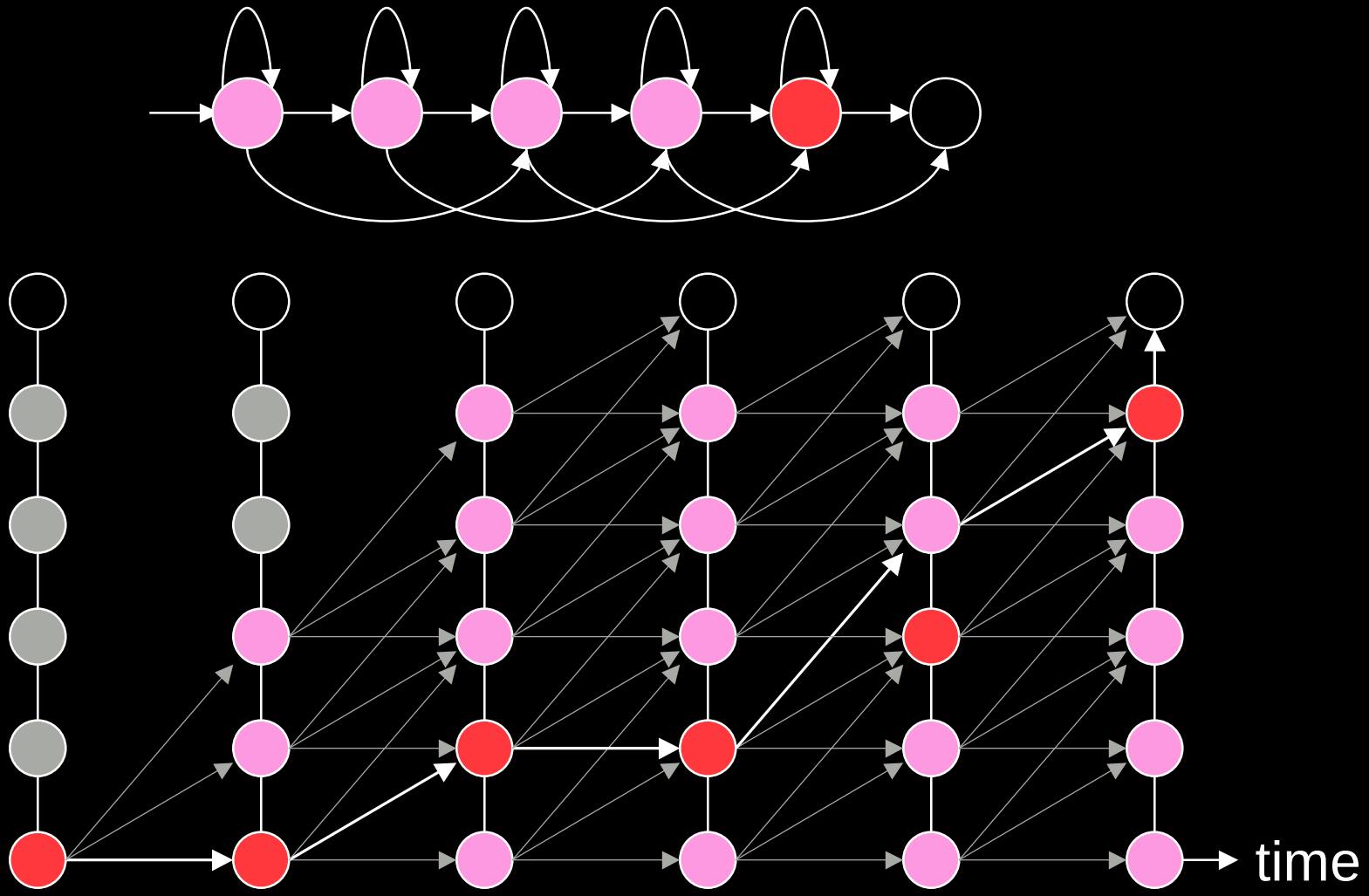
time

Sphinx Speech Group, CMU-SCS (rkm@cs.cmu.edu)

Viterbi Search (contd.)



Viterbi Search (contd.)

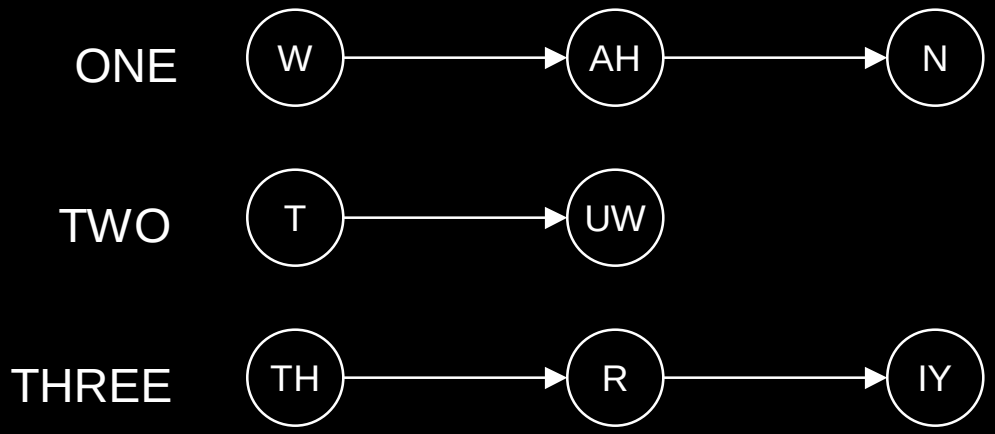


Viterbi Search Summary

- *Instantaneous score*: how well a given HMM state matches the speech input at a given time frame
- *Path*: A sequence of HMM states traversed during a given segment of input speech
- *Path-score*: Product of instantaneous scores and state transition probabilities corresponding to a given path
- *Viterbi search*: An efficient lattice structure and algorithm for computing the best path score for a given segment of input speech

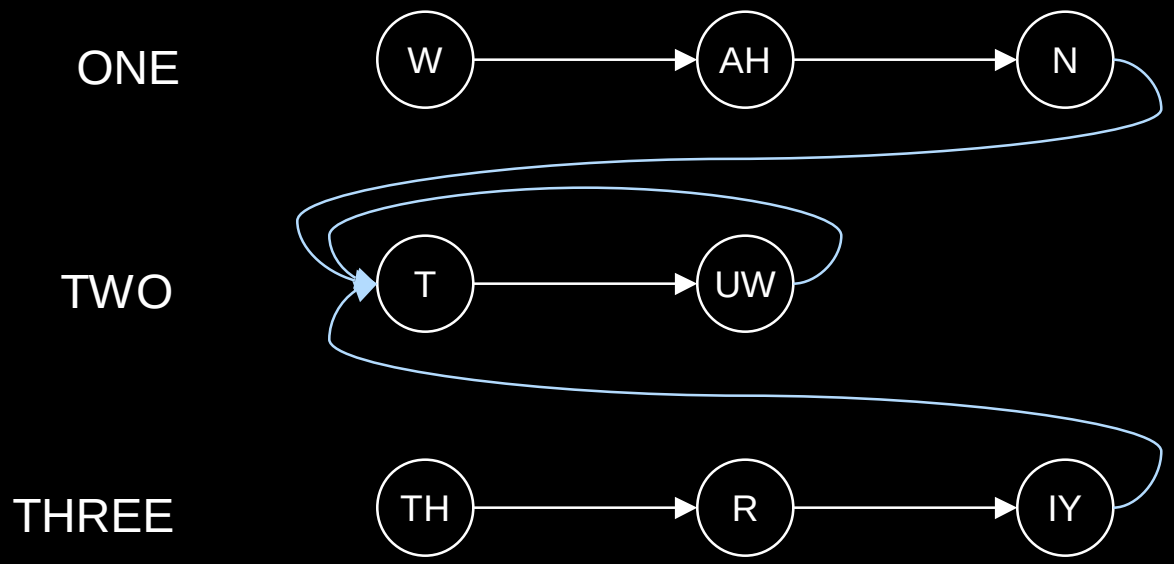
Single Word Recognition

- **Search all words in parallel**
  - **Initialize start state of every word with path-score 1.0**
  - **For each frame of input speech:**
    - **Update path scores within each HMM**
    - **Propagate exit state score from one HMM to initial state of its successor (using Viterbi criterion)**
  - **Select word with best exit state path-score**



Continuous Speech Recognition

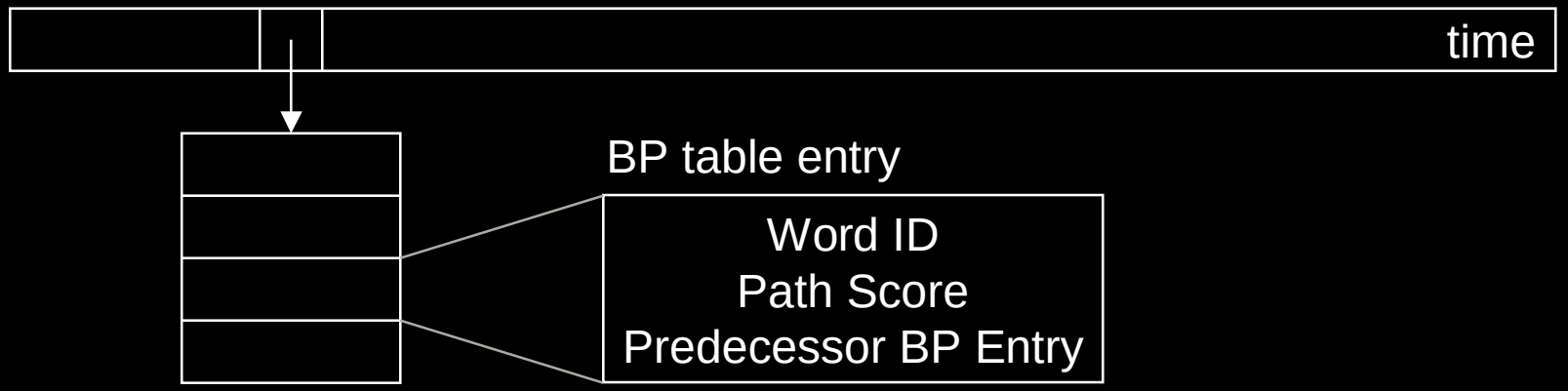
- **Add null transitions from word ends to beginnings:**



- **Apply Viterbi search algorithm to the modified network**
- **Q: How to recover the recognized word sequence?**

The Backpointer Table

- **Each word exit recorded in the BP table:**



- **Upon transitioning from an exited word A to another B:**
  - **Inject pointer to BP table entry for A into start state of B. (This identifies the predecessor of B.)**
  - **Propagate these pointers along with path-scores during Viterbi search**
- **At end of utterance, identify best exited word and trace back using predecessor pointers**

The Backpointer Table (contd.)

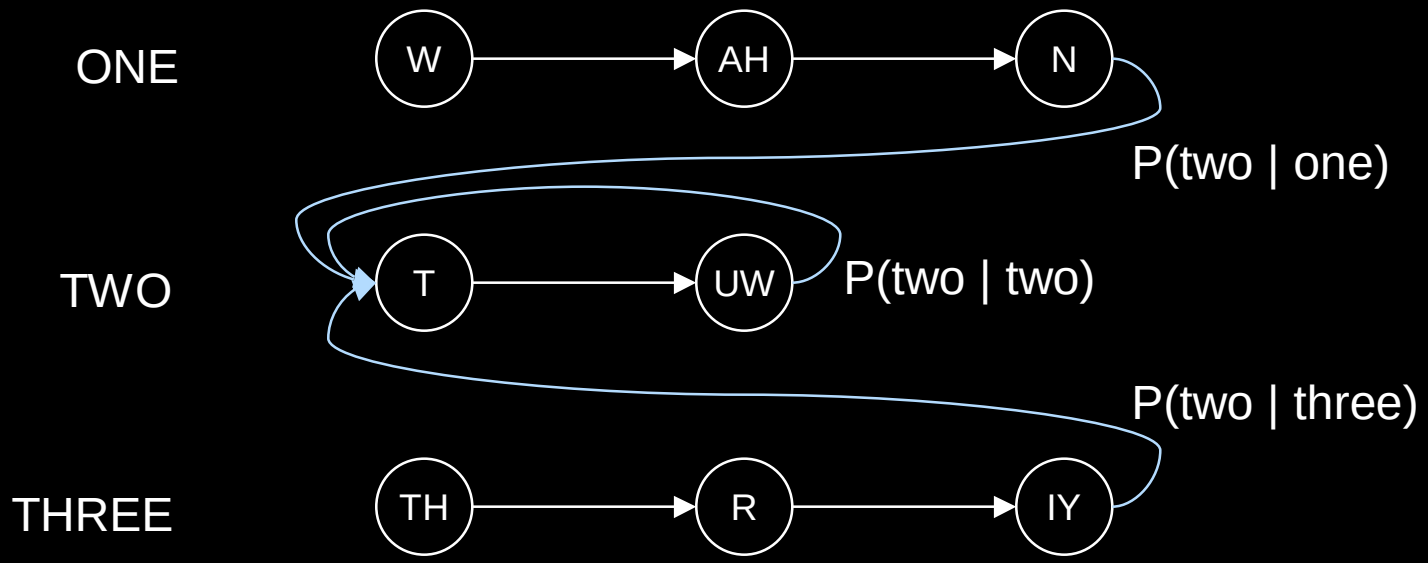
- Some additional information available from BP table:
  - All candidate words recognized during recognition
  - Word segmentations
  - Word segment acoustic scores
  - "Lattice density": No. of competing word hypotheses at any instant
- Useful for postprocessing steps:
  - Lattice rescoring
  - N-best list generation
  - Confidence measures

Beam Search (Pruning)

- Exhaustive search over large vocabulary too expensive, and unnecessary
- Use a "beam" to "prune" the set of active HMMs:
  - At start of each frame, find best available path-score S
  - Use a scale-factor f (< 1.0) to set a pruning threshold $T = S*f$
  - Deactivate an HMM if no state in it has path score >= T
- Effect: No. of active HMMs larger if no clear frontrunner
- Two kinds of beams:
  - To control active set of HMMs
    - No. of active HMMs per frame typically 10-20% of total space
  - To control word exits taken (and recorded in BP table)
    - No. of words exited typically 10-20 per frame
- Recognition accuracy essentially unaffected

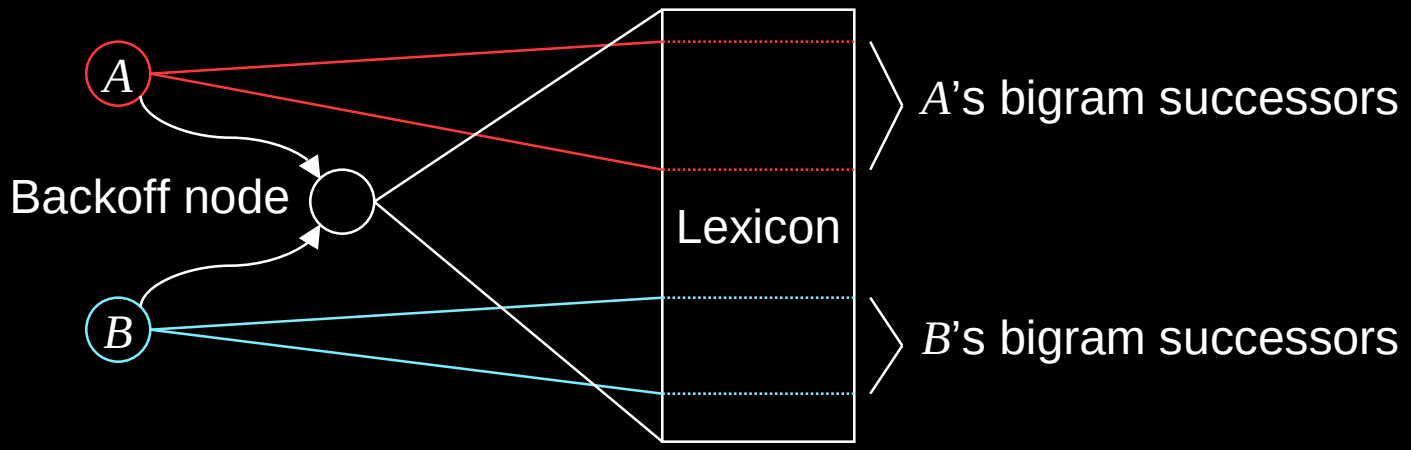
Incorporating a Language Model

- **Language models essential for recognition accuracy**
  - **Reduce word error rate by an order of magnitude**
  - **Reduce active search space significantly**
- **Implementation: associate LM probabilities with transitions between words. E.g.:**



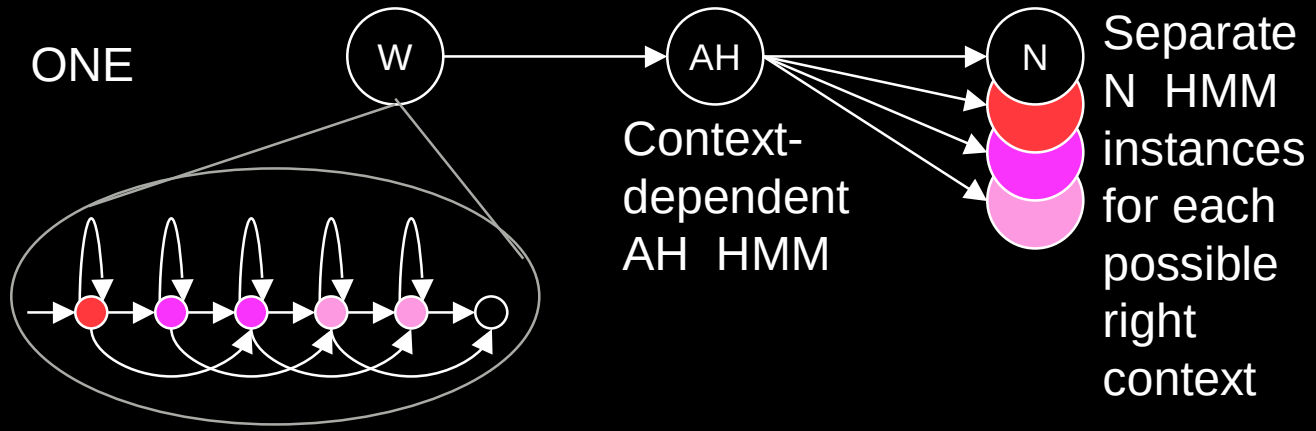
Bigram Backoff Language Model

- **Two issues with large vocabulary bigram LMs:**
  - **With vocabulary size V and N word exits per frame, NxV cross-word transitions per frame**
  - **Bigram probabilities very sparse; mostly "backoff" to unigrams**
- **Optimize cross-word transitions using "backoff node":**
  - **Viterbi decision at backoff node selects single-best predecessor**



Cross-Word Triphone Modeling

- **Sphinx uses "triphone" or "phoneme-in-context" HMMs**



Multiplexed W HMM; inherited left context propagated along with path-scores, and dynamically modifies the state model (Replication is too expensive)

- **Cross-word transitions use appropriate exit-model, and inject left-context into entry state**

Sphinx-3 Search Algorithm

```
initialize start state of <S> with path-score = 1;
for each frame of input speech {
    evaluate all active HMMs; find best path-score, pruning thresholds;
    for each active HMM {
        if above pruning threshold {
            activate HMM for next frame;
            transition to and activate successor HMM within word, if any
            if word-final HMM and above word-pruning threshold
                record word-exit in BP table;
        }
    }
    transition from words exited into initial state of entire lexicon (using the
        LM), and activate HMMs entered;
}
find final </S>  BP table entry and back-trace through table to retrieve result;
```

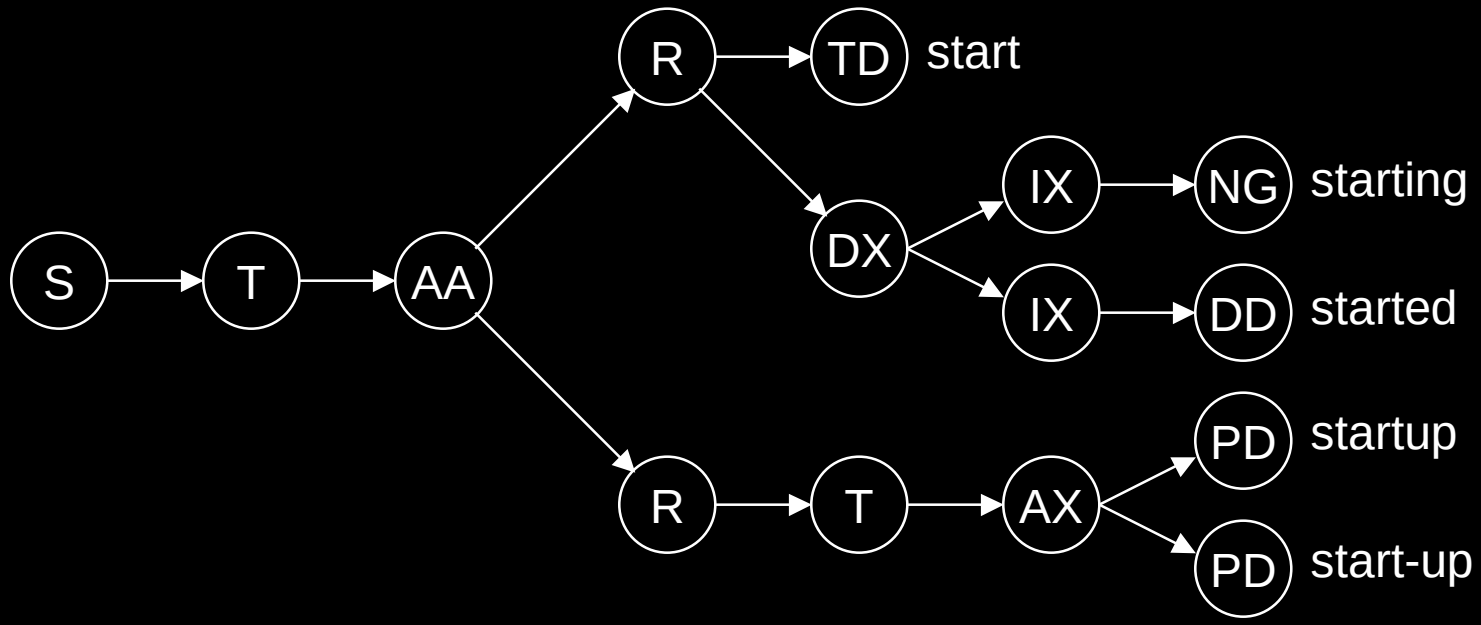
Lextree Search: Motivation

- **Most active HMMs are word-initial models, decaying rapidly subsequently**
  - **On 60K-word Hub-4 task, 55% of active HMMs are word-initial**
  - **(Same reason for handling left/right contexts differently.)**
- **But, no. of distinct word-initial model types much fewer:**

START	S-T-AA-R-TD
STARTING	S-T-AA-R-DX-IX-NG
STARTED	S-T-AA-R-DX-IX-DD
STARTUP	S-T-AA-R-T-AX-PD
START-UP	S-T-AA-R-T-AX-PD

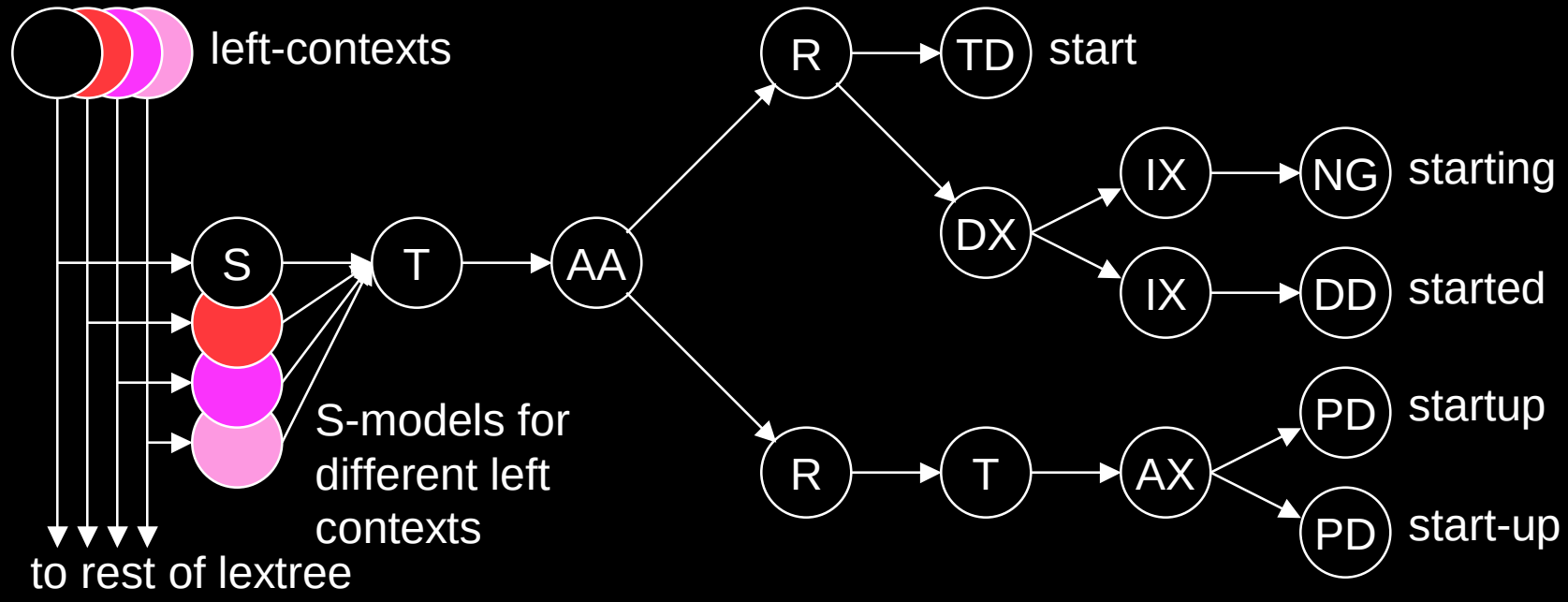
- **Use a "prefix-tree" structure to maximize sharing among words**

Lextree Structure in Sphinx-3.2



- Nodes shared if triphone “State-Sequence ID” (SSID) identical
- Leaf (word-final) nodes not shared
- In 60K-word BN task, word-initial models reduced ~50x

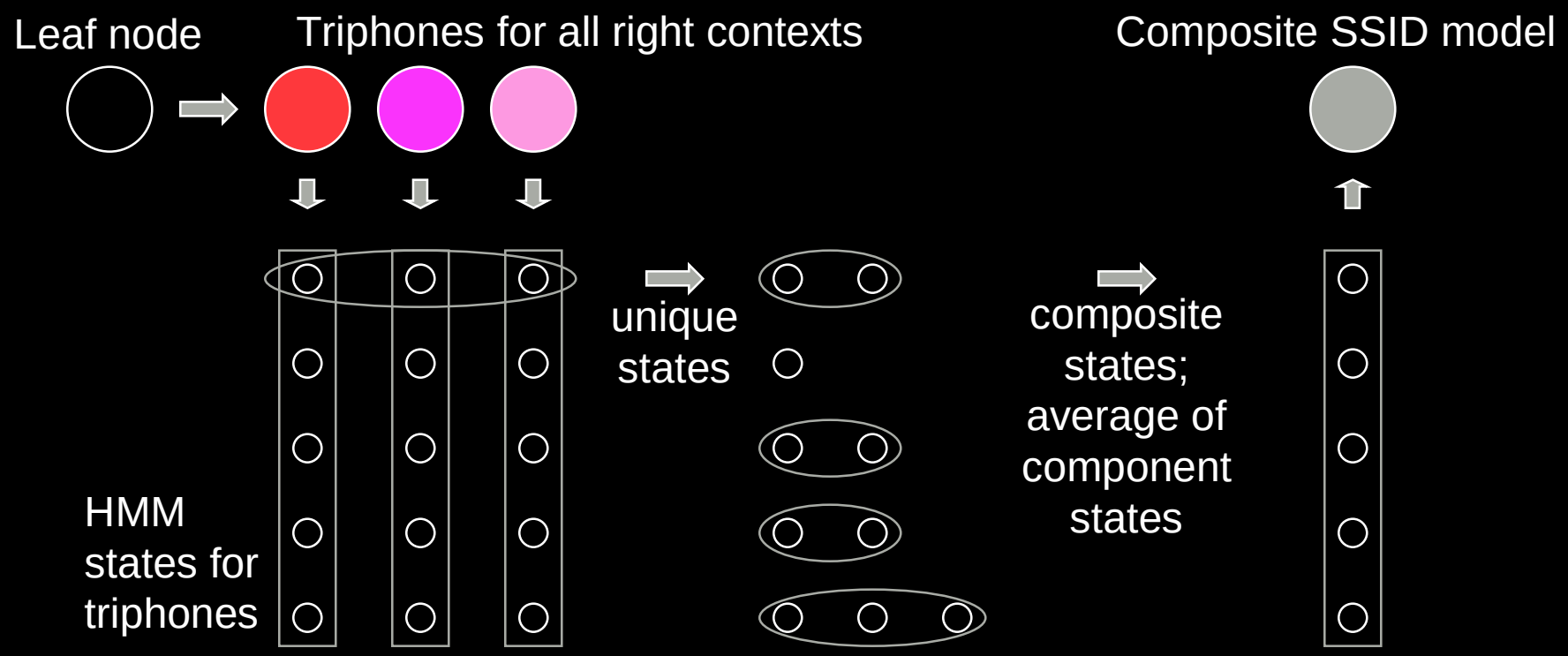
Cross-Word Triphones (left context)



- **Root nodes replicated for left context**
  - **Again, nodes shared if SSIDs identical**
  - **During search, very few distinct incoming left-contexts at any time; so only very few copies activated**

Cross-Word Triphones (right context)

- **Leaf nodes use "composite" SSID models**
  - **Simplifies lextree and backpointer table implementation**
  - **Simplifies cross-word transitions implementation**

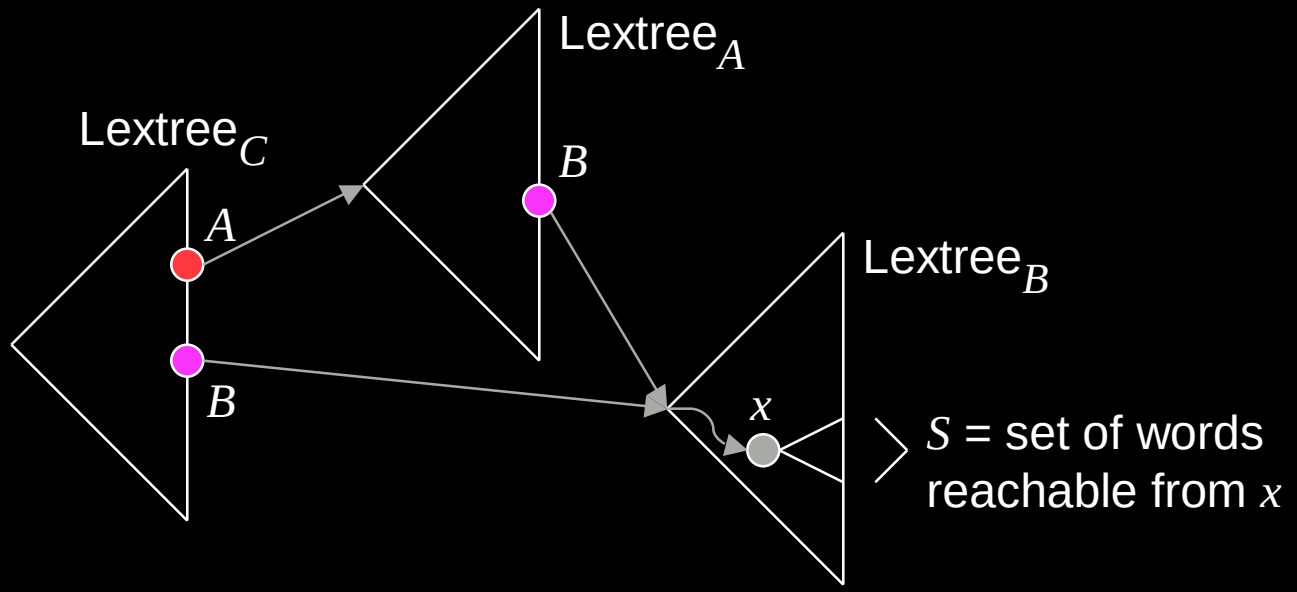


Lextree Search: LM Integration

- Problem: LM probabilities cannot be determined upon transition to lextree root nodes
  - Root nodes shared among several unrelated words
- Several solutions possible:
  - Incremental evaluation, using composite LM scores
    - Lextree replication (Ney, Antoniol)
    - Rescoring at every node (BBN)
  - Post-tree evaluation (Sphinx-II)

LM Integration: Lextree Replication

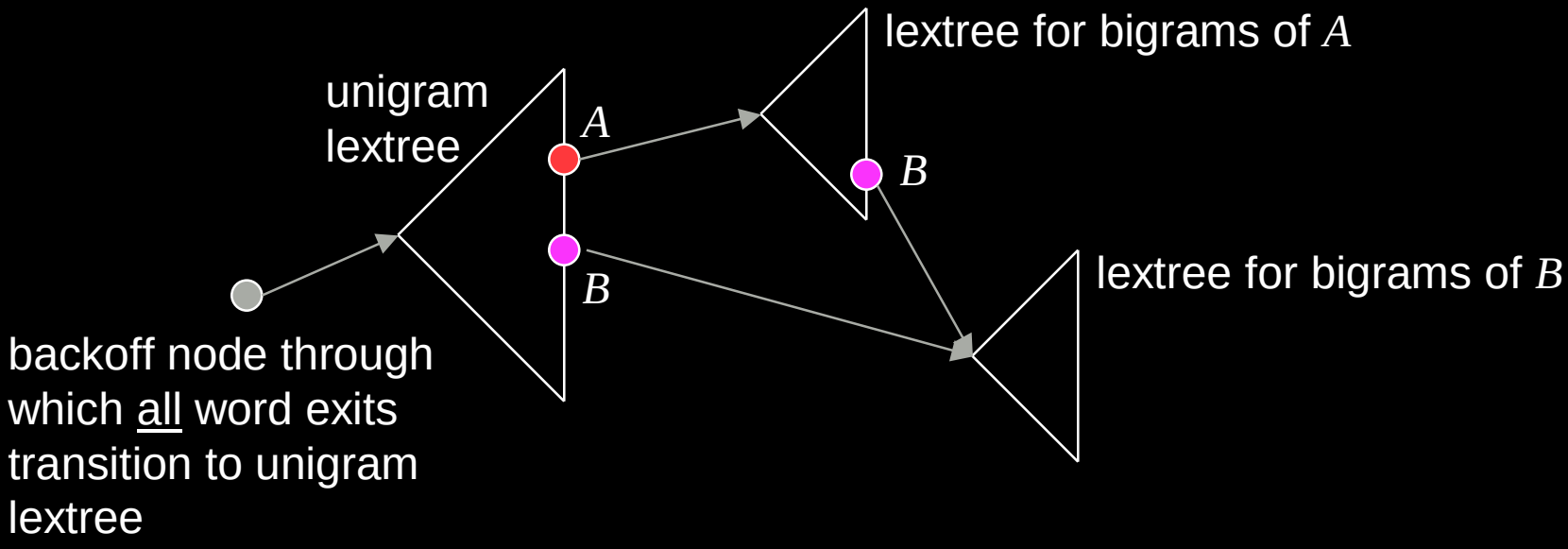
- **Incremental LM score accumulation; e.g. (bigram LM):**



- **Large computation and memory requirements**
- **Overhead for dynamic lextree creation/destruction**

Lextree Copies With Explicit Backoff

- **Again, incremental LM score accumulation:**

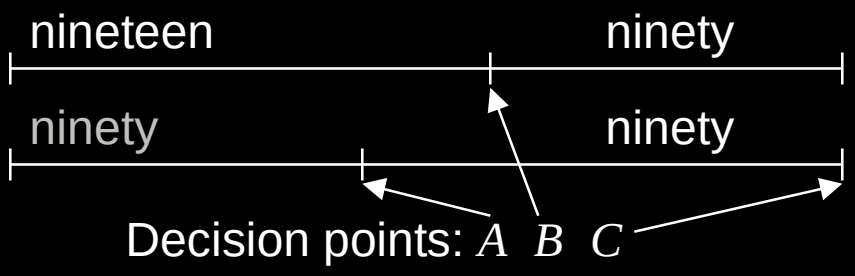


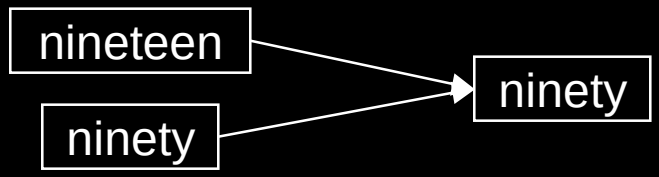
- **Still, large computation/memory requirements and overhead for dynamic lextree maintenance**
- **Multiple LM transitions between some word pairs**

Post-Lextree LM Evaluation (Sphinx-II)

- Single lextree
  - Null transitions from leaf nodes back to root nodes
- No LM score upon transition into root or non-leaf node of lextree
- If reached a leaf node for word W:
  - Find all possible LM histories of W (from BP table)
  - Find LM scores for W w.r.t. each LM history
  - Choose best resulting path-score for W
- Drawbacks:
  - Inexact acoustic scores
    - Root node evaluated w.r.t. a single left context, but resulting score used w.r.t. all histories (with possibly different left contexts)
  - Impoverished word segmentations

Word Segmentation Problem



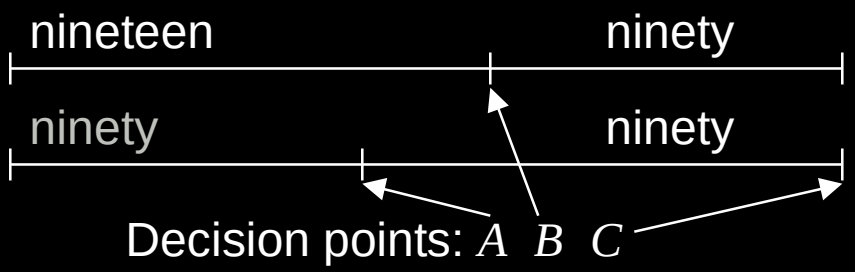


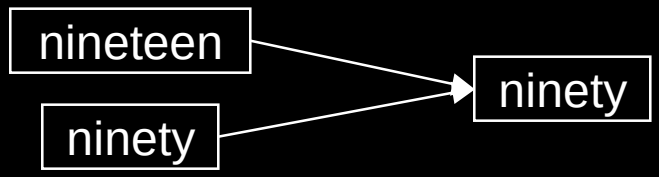
Assume P (*ninety* | *nineteen*) > P (*ninety* | *ninety*).

Q: Which transition wins?

- **Flat lexicon (separate model per word):**
  - **At A: word *ninety* entered with LM score P (*ninety* | *ninety*)**
  - **At B: word *ninety* entered with P (*ninety* | *nineteen*)**
    - **Since the latter is much better, it prevails over the former**
  - **Result: correct recognition, and segmentation for *ninety***

Word Segmentation Problem (contd.)





Assume P (*ninety* | *nineteen*) > P (*ninety* | *ninety*).

- **Tree lexicon:**
  - **At A: root node for *ninety* entered without any LM score**
  - **At B: Attempt to enter root node for *ninety* again**
    - **Transition may or may not succeed (no LM score used)**
  - **At C: obtain LM score for *ninety* w.r.t. all predecessors**
    - **If transition at B failed, the only candidate predecessor is *ninety*; result: incorrect segmentation for *ninety*(2), incorrect recognition**

Lextree-LM Integration in Sphinx-3.2

- Post-lextree LM scoring (as above); however:
- Limited, static lextree replication
  - Limits memory requirements
  - No dynamic lextree management overhead
- Transitions into lextrees staggered across time:
  - At any time, only one lextree entered
    - "-epl" (entries per lextree) parameter: block of frames one lextree entered, before switching to next
  - More word segmentations (start times) survive
- Full LM histories; if reached a leaf node for word W:
  - Find all possible LM histories of W (from BP table)
  - Include LM scores for W w.r.t. each LM history
    - Create a separate BP table entry for each resulting history

Pruning in Sphinx-3.2

- Pure beam-pruning has long-tailed distribution of active HMMs/frame
- Absolute pruning to control worst-case performance:
  - Max. active HMMs per frame
    - Implemented approximately, using “histogram pruning” (avoids expensive sorting step)
  - Max. unique words exiting per frame
  - Max. LM histories saved in BP table per frame
  - Word error rate unaffected
- Additional beam for lextree-internal, cross-HMM transitions
- Unigram “lookahead” scores used in lextree for yet more pruning

Sphinx-3.2 Performance

- **1997 BN eval set, excluding F2 (telephone speech)**
- **6K tied-state CHMM, 20 density/state model (1997)**
- **60K vocab, trigram LM**

config	pruning param. wd/f	bp/f	hmm/f	WER	WER %incr	active hmm/f	bp/f	xRT srch	xRT total	P-III MHz
flat-wb	--	--	--	26.8	--	78.0K	25*	42.0	49.0	400
flat-nb	--	--	--	29.3	9.2	12.9K	9*	12.2	19.2	400
trees:3	20	--	--	28.2	5.2	6.8K	42	2.4	7.4	400
3	20	100	10K	28.2	5.2	5.8K	35	2.2	7.2	400
3	20	50	10K	28.3	5.4	5.8K	25	2.0	7.1	400
2	20	100	10K	28.7	7.2	5.2K	34	1.4	5.9	450
1	20	100	10K	29.3	9.3	3.4K	32	1.0	6.0	400

Sphinx Speech Group, CMU-SCS (rkm@cs.cmu.edu)

Sphinx-3.2 Performance (contd.)

- **1998 BN Eval set**
- **5K tied-state CHMM, 32 density/state model (1998)**
- **60K vocab, trigram LM**

	pruning param.				WER	active		xRT		P-III
config	wd/f	bp/f	hmm/f	WER	%incr	hmm/f	bp/f	srch	total	MHz
flat-wb	10	--	--	21.5	--	103K	9*	42.7	49.3	425*
trees:3	20	--	--	22.0	2.3	10.8K	52	3.1	8.5	450
3	20	--	20K	22.0	2.3	9.7K	52	3.1	8.4	450
3	20	100	20K	22.0	2.3	9.7K	41	3.3	9.2	400
3	20	--	10K	22.1	2.8	7.9K	52	3.3	9.2	400
3	20	100	10K	22.1	2.8	7.9K	40	2.6	7.9	450
3	20	50	10K	22.1	2.8	7.9K	29	2.8	8.6	400
3	20	30	10K	22.1	2.8	7.9K	21	2.6	8.5	400
2	20	100	10K	22.4	4.2	7.2K	39	2.0	7.3	450
1	20	100	10K	23.2	7.9	5.1K	38	1.4	6.6	450

Sphinx Speech Group, CMU-SCS (rkm@cs.cmu.edu)

Sphinx-3.2 Performance (contd.)

- **Effect of absolute pruning parameters (1998 BN):**
  - **(Per frame) computation stats for each utterance**
  - **Distribution of stats over entire test set (375 utts)**
- **Absolute pruning highly effective in controlling variance in computational cost**

		Active HMM/fr			BPtbl entries/fr			Srch. xRT		
hmm/f	bp/f	Mean	Max	StDev.	Mean	Max	StDev.	Mean	Max	StDev.
--	--	11.2K	46.3K	5.5K	53	220	24.6	3.2	33.5	2.7
20K	--	10.0K	29.1K	3.8K	53	220	24.5	3.2	31.5	2.6
20K	100	10.0K	29.1K	3.8K	41	90	12.3	3.4	21.4	2.2
10K	--	8.1K	17.1K	2.2K	53	216	24.4	3.4	31.6	2.6
10K	100	8.1K	17.1K	2.2K	41	90	12.4	2.7	13.7	1.4
10K	50	8.1K	17.1K	2.2K	30	47	5.8	2.9	11.6	1.3
10K	30	8.1K	17.3K	2.2K	21	29	2.9	2.7	9.7	1.1

Conclusions

- 10x CHMM system available (thanks to P-!!!)
- With lextree implementation, only about 1-2% of total HMMs active per frame
  - Order of magnitude fewer compared to flat lexicon search
- Lextree replication improves WER noticeably
- Absolute pruning parameters improve worst-case behavior significantly, without penalizing accuracy
  - When active search space grows beyond some threshold, no hope of correct recognition anyway

What Next?

- Tree lexicon still not as accurate as flat lexicon baseline
  - Residual word segmentation problems?
    - Try lextree replication?
  - Use of composite SSID model at leaf nodes?
  - Parameters not close to optimal?
- HMM state acoustic score computation now dominant
  - Back to efficient Gaussian selection/computation