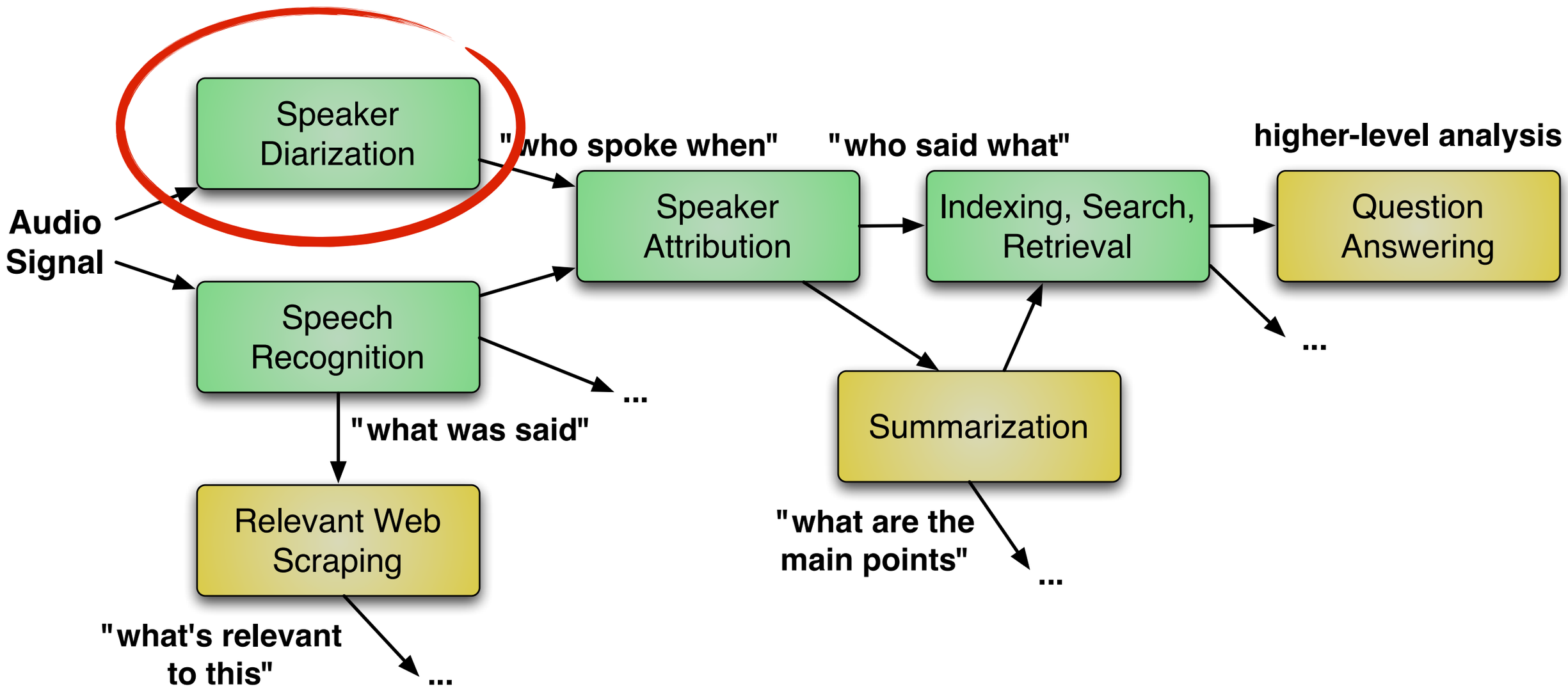
PARALLEL COMPUTING LABORATORY

The Meeting Diarist: An Example of Qualitative Improvements through Parallelization

Dr. Gerald Friedland
International Computer Science Institute (ICSI)
Work mainly together with:
Ekatarina Gonina, Henry Cook, Adam Janin, Armando Fox

The Meeting Diarist

Components of the Meeting Diarist

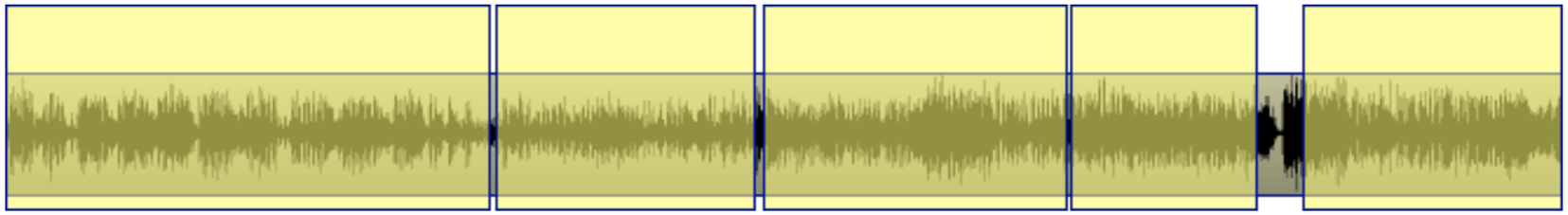


Overall Structure: Pipe & Filter

Speaker Diarization – Definition

Audiotrack:

Segmentation:

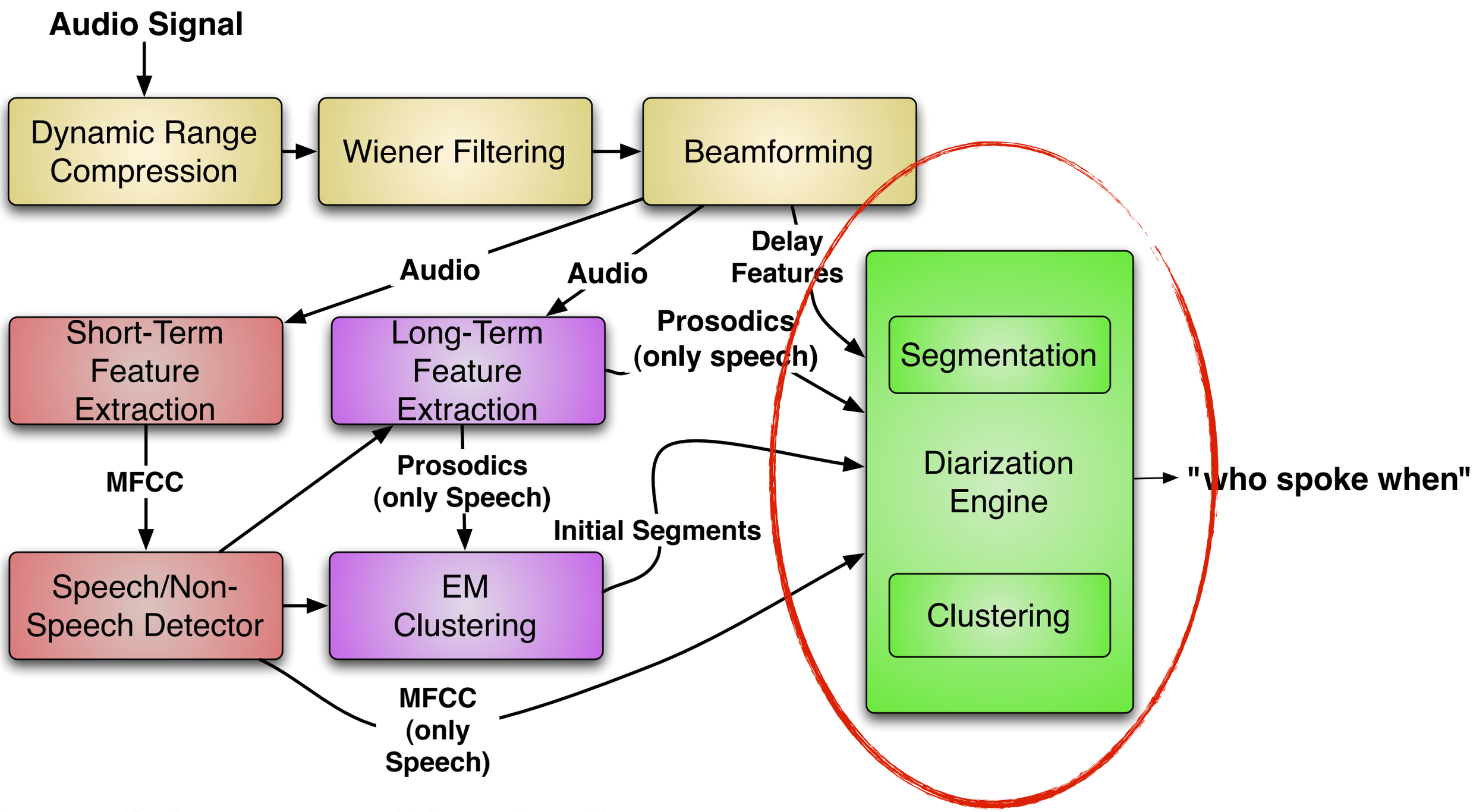


Clustering:

Estimate “who spoke when” with no prior knowledge of speakers, #of speakers, words, or language spoken.

Speaker Diarization – Overview

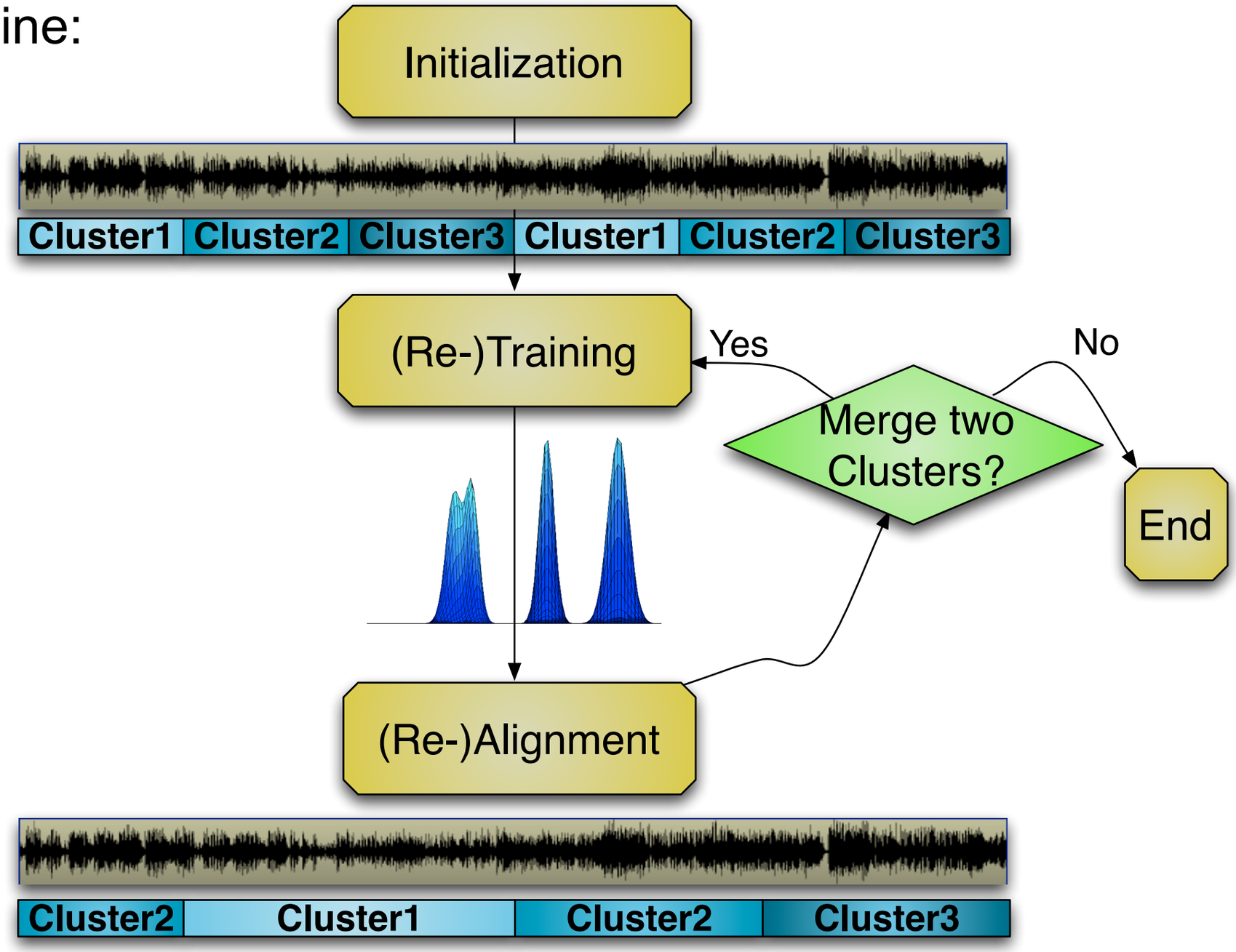
Processing Chain:



Overall Structure: Pipe & Filter

Speaker Diarization: Core Algorithm

Algorithm outline:



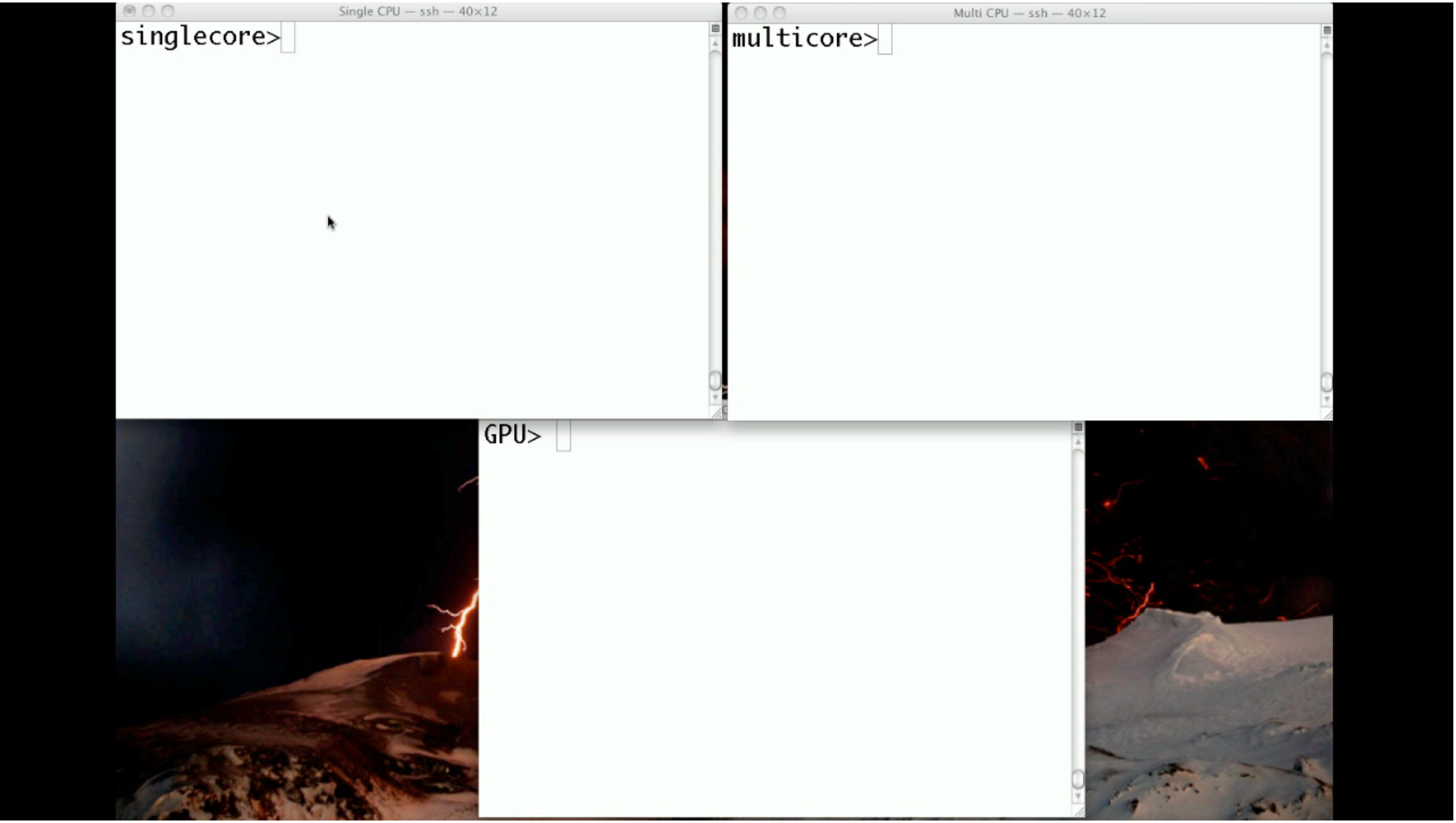
- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed

Parallelization of Diarization

Five versions (so far):

- Initial code (2006): 0.333 x realtime (i.e., 1h audio = 3h processing)
- Serially optimized (2008): 1.5 x realtime
- Parlab retreat summer 2010: Multicore+GPU parallelization: 14.3 x realtime
- Parlab retreat winter 2011: GPU-only parallelization 250 x realtime (i.e., 1h audio = 14.4sec processing)
- Parlab retreat summer 2011: SEJITized!

Speaker Diarization on GPU

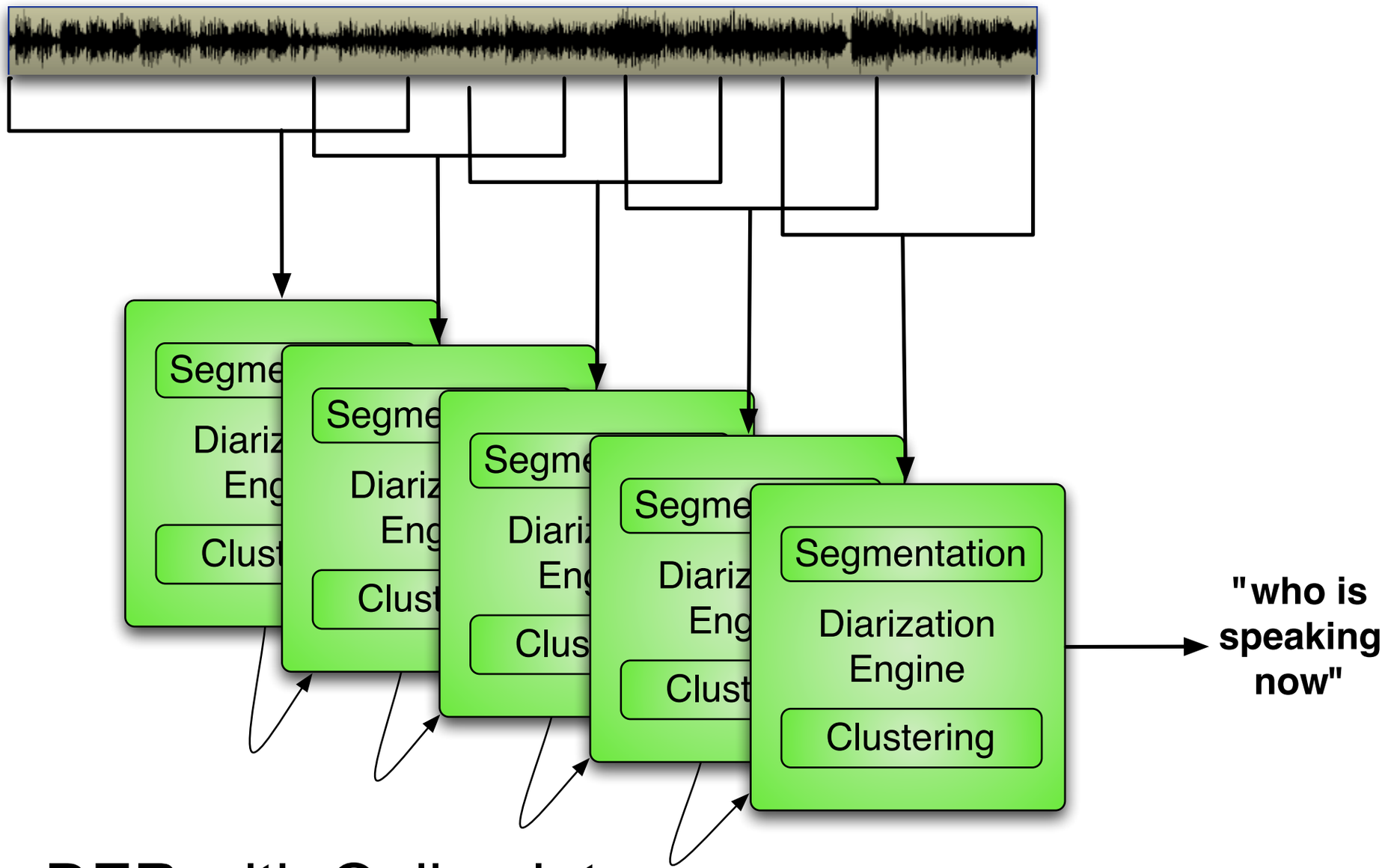


Now it's fast: So what?

- Algorithm can be tuned to leverage the "spare time" -> Higher accuracy
- Algorithm can be tuned using much more data (e.g., 50k TrecVID videos) -> Increased robustness
- AHC with GMMs interesting for other applications as well
- Now online = offline!

Slide from Summer Retreat 2010

Imagine an offline Speaker Diarization System that responds in > 1000xRT (i.e. diarizes 1000s in 1s)...



-> Offline DER with Online latency.
Convergence at 350s per 2.5s = only 140 x realtime!

Impact!

Noulas, A.K. and Krose, B.J.A, “On-line Multi-Modal speaker Diarization”, International Conference on Multi-modal Interfaces, ICMI’07, 350–357, 2007.

Markov, K. and Nakamura, S., “Never-Ending Learning System for On-line Speaker diarization”, in Proc. IEEE ASRU’07, 699–704, Kyoto, Japan, 2007.

Vinyals, O. and Friedland, G., “Towards semantic analysis of conversations: A system for the live identification of speakers in meetings”, Proc. IEEE ICSC’08, 426–431, Santa Clara, CA, 2008.

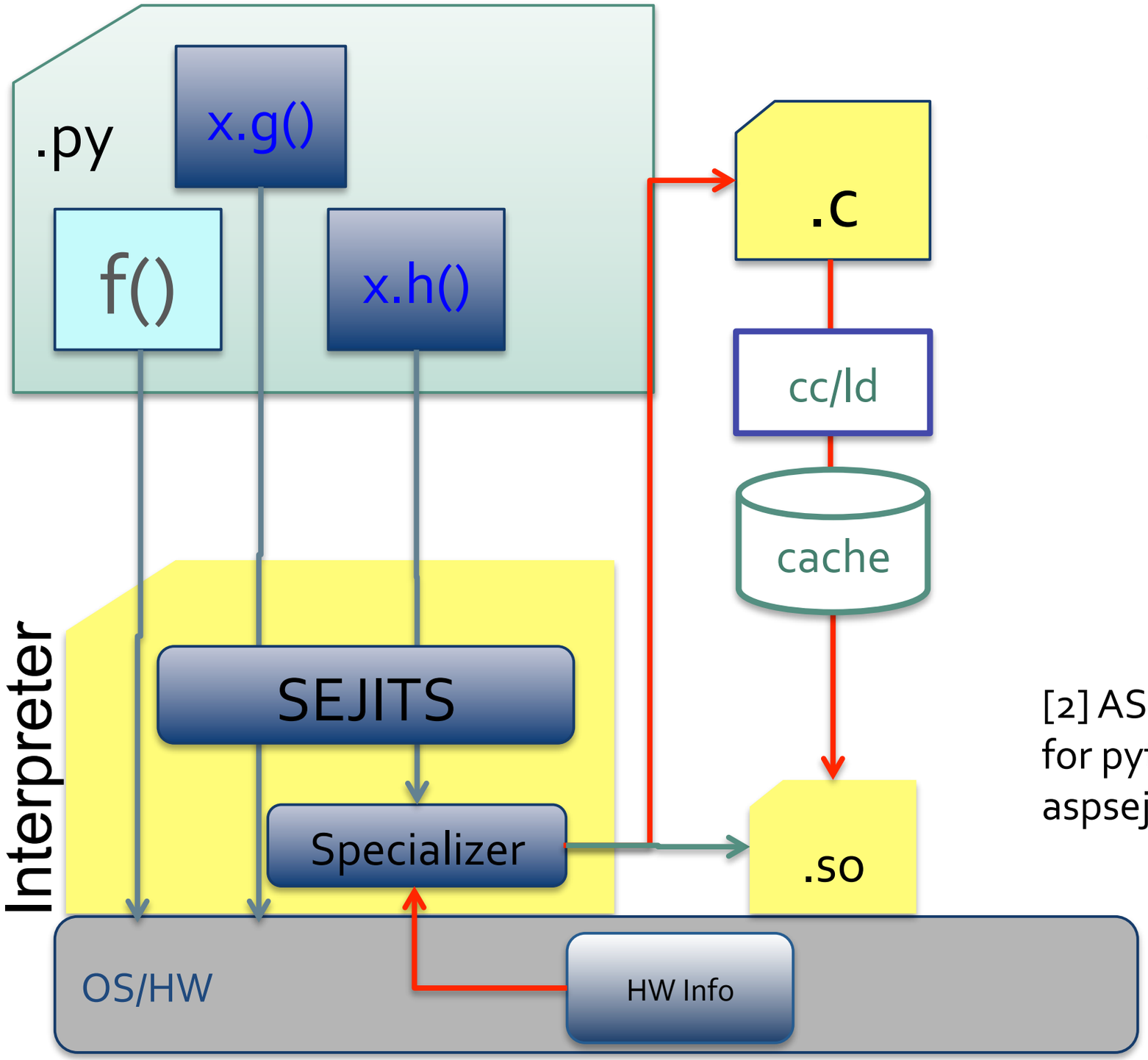
Hung, H. and Friedland, G., “Towards Audio-Visual On-line Diarization of Participants In Group Meetings”,Workshop on Multicamera and multi-modal Sensor Fusion, M2SFA2, Proeedings of ECCV, 2008.

Schmalenstroeer, M. et al, “Fusing audio and Video Information for Online Speaker Diarization”, in Proc. Interspeech’09, 1163–1166, Brighton, UK, 2009.

Limits of the Approach

- "Lots of brain-power" needed as it's an efficiency layer solution
- Highly specialized: Only works on CUDA
- Feature extraction, speech/non-speech detection now bottleneck -> Needs to be addressed before solution is practical

Selective Embedded JIT Specialization (SEJITS)



ASP – A SEJITS for Python [1]

[2] ASP: A SEJITS implementation for python. http://aspsejits.pbworks.com/

SEJITized Diarization

```
def AHC(self):

  # Get the events, divide them into an initial k clusters and train each GMM on a cluster
      per_cluster = self.N/self.init_num_clusters
      init_training = zip(self.gmm_list,np.vsplit(self.X, range(per_cluster, self.N, per_cluster)))
      for g, x in init_training:
          g.train(x)

  # Perform hierarchical agglomeration based on BIC scores
      best_BIC_score = 1.0
      while (best_BIC_score > 0 and len(self.gmm_list) > 1):

          num_clusters = len(self.gmm_list)

          # Resegment data based on likelihood scoring
          likelihoods = self.gmm_list[0].score(self.X)
          for g in self.gmm_list[1:]:
              likelihoods = np.column_stack((likelihoods, g.score  (self.X)))
          most_likely = likelihoods.argmax(axis=1)

          # Across 2.5 secs of observations, vote on which cluster they should be associated with
          split_events = split_events_based_on_votes(most_likely, self.X)

          for g, data in split_events:
                g.train(data)

          # Score all pairs of GMMs using BIC
          best_merged_gmm = None
          best_BIC_score = 0.0
          merged_tuple = None

          for gmm1idx in range(len(iter_bic_list)):
              for gmm2idx in range(gmm1idx+1, len(iter_bic_list)):
                  g1, d1 = iter_bic_list[gmm1idx]
                  g2, d2 = iter_bic_list[gmm2idx]
                  score = 0.0
                  new_gmm, score = compute_distance_BIC(g1, g2, np.concatenate((d1, d2)))
                  if score > best_BIC_score:
                      best_merged_gmm = new_gmm
                      merged_tuple = (g1, g2)
                      best_BIC_score = score

          # Merge the winning candidate pair
          if best_BIC_score > 0.0:
              self.gmm_list.remove(merged_tuple[0])
              self.gmm_list.remove(merged_tuple[1])
              self.gmm_list.append(best_merged_gmm)
```

SEJITized Diarization

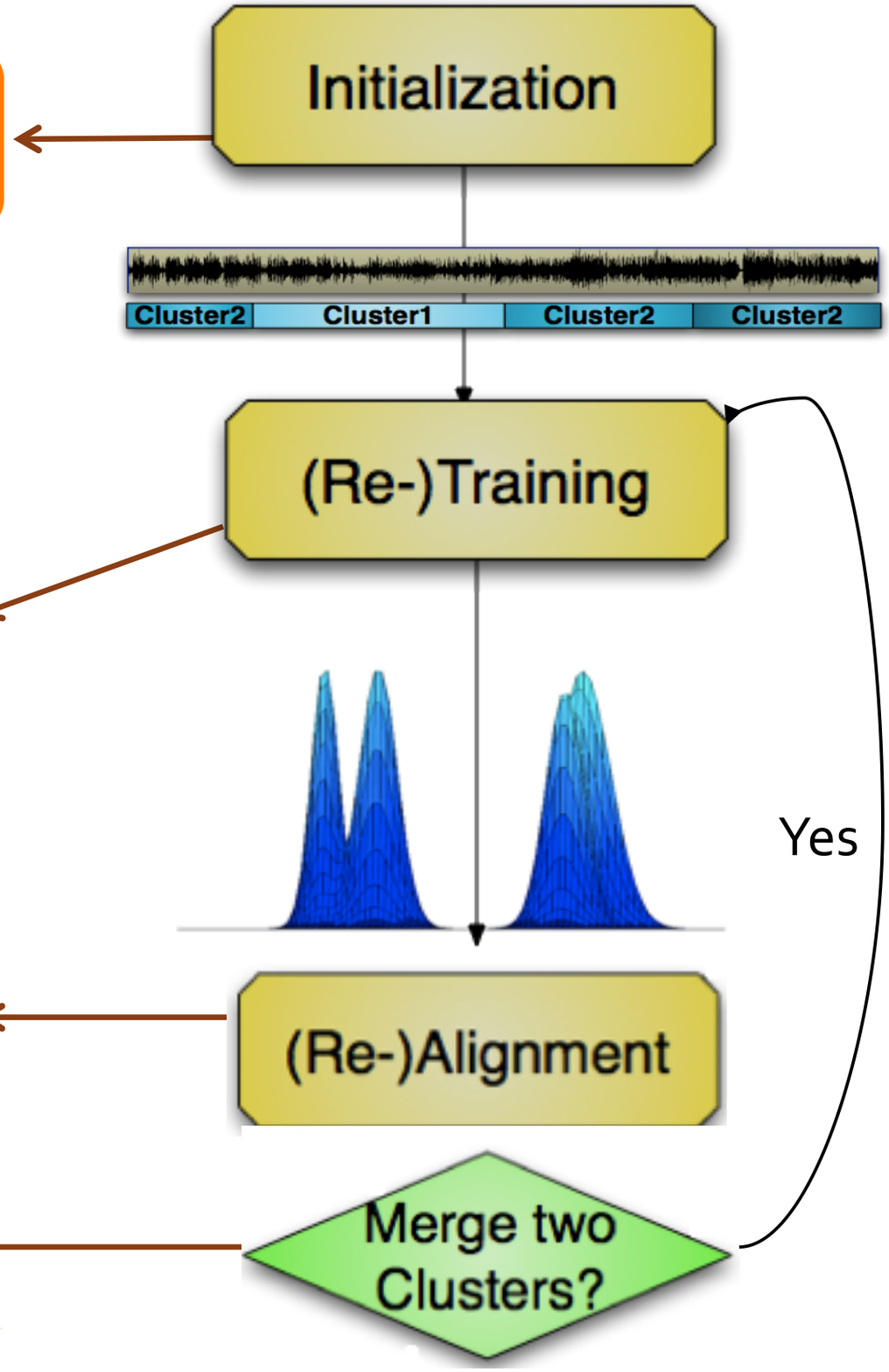
```
def AHC(self):

    # Ge                                        's and train each GMM on a cluster
    per_
    init                                        ıge(per_cluster, self.N, per_cluster)))
    for
        g.train(x)

    # Perform hierarchical agglomeration based on BIC scores
    best_BIC_score = 1.0
    while (best_BIC_score > 0 and len(self.gmm_list) > 1):

        num_clusters = len(self.gmm_list)

        # Resegment data based on likelihood scoring
        likelihoods = self.gmm_list[0].score(self.X)
        for g in self.gmm_list[1:]:
            likelihoods = np.column_stack((likelihoods, g.score  (self.X)))
        most_likely = likelihoods.argmax(axis=1)

        # Across 2.5 secs of observations, vote on which cluster they should be associated with
        split_                          _on_votes(most_likely, self.X)

        for g,
            g.train(x)

        # Score all pairs of GMMs using BIC
        best_merged_gmm = None
        best_BIC_score = 0.0
        merged_tuple = None

        for gmm1idx in range(len(iter_bic_list)):
            for gmm2idx in range(gmm1idx+1, len(iter_bic_list)):
                g1, d1 = iter_bic_list[gmm1idx]
                g2, d2 = iter_bic_list[gmm2idx]
                score = 0.0
                new_gmm, score = com                    ).concatenate((d1, d2)))
                if score > best_BIC_
                    best_merged_gmm = new_gmm
                    merged_tuple = (g1, g2)
                    best_BIC_score = score

        # Merge the winning candidate pair
        if best_BIC_score > 0.0:
            self.gmm_list.remove(merged_tuple[0])
            self.gmm_list.remove(merged_tuple[1])
            self.gmm_list.append(best_merged_gmm)
```

new_gmm_list(M,D)

g.train(x)

g.train(x)

g.train(x)



Conclusion

Parallelization allows for more than incremental speed-up. It introduces new possibilities on how to approach a problem.

- higher-level languages and the speed-up make experimenting easier
- the algorithm can iterate more often
- due to the speed-up, task-specific optimization can be avoided

Advertisement

CS298-53 (26862): Acoustic Methods for Video Analysis

Fridays, 1:30-3pm, 320 SODA

More information:
http://www.icsi.berkeley.edu/~fractor/fall2011/

Last Slide

Thank You!

Questions?

Contact:
Dr. Gerald Friedland
International Computer Science Insitute
email: fractor@icsi.berkeley.edu
web: http://www.gerald-friedland.org