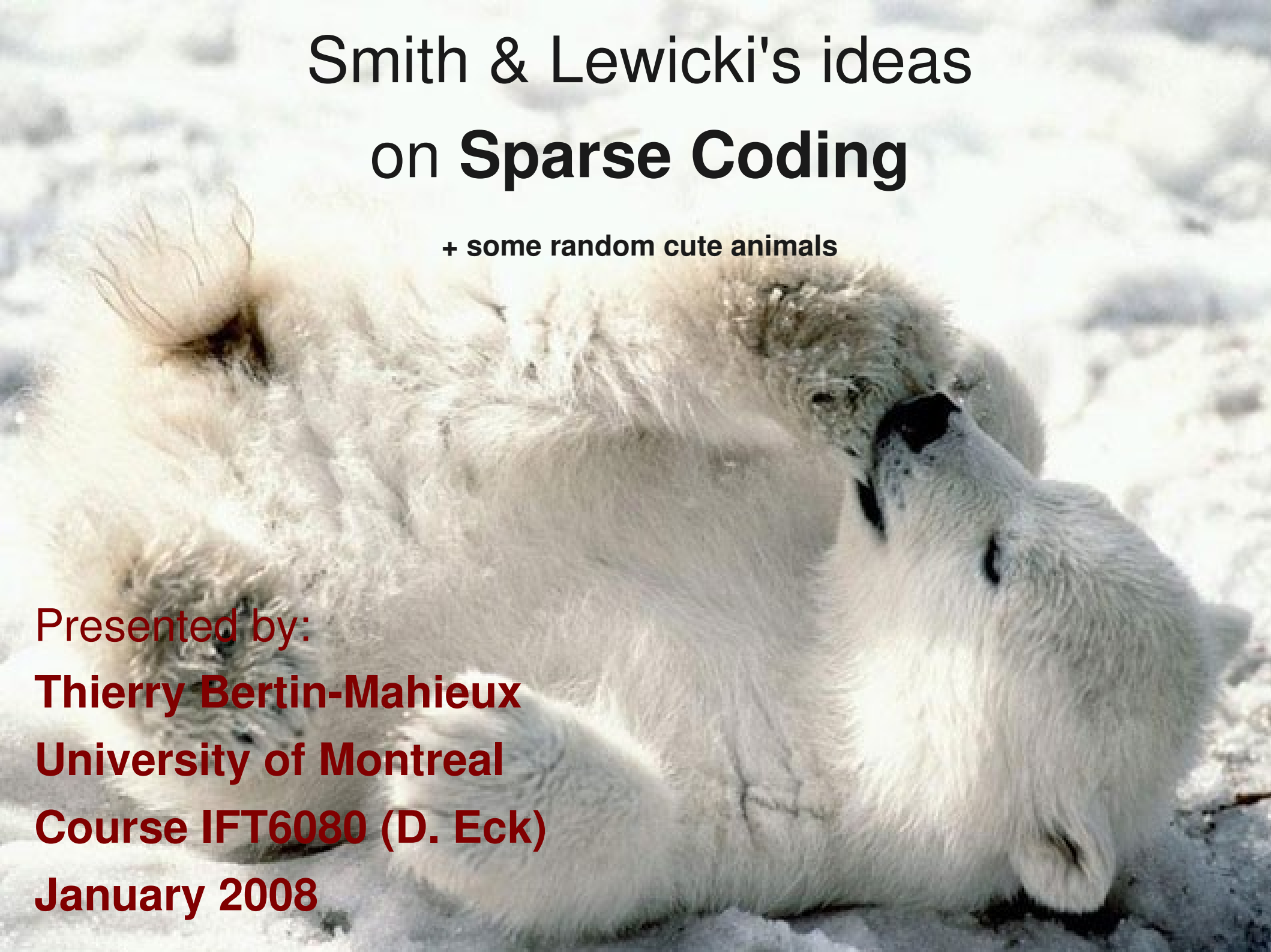
Smith & Lewicki's ideas on **Sparse Coding**

+ some random cute animals



Presented by:

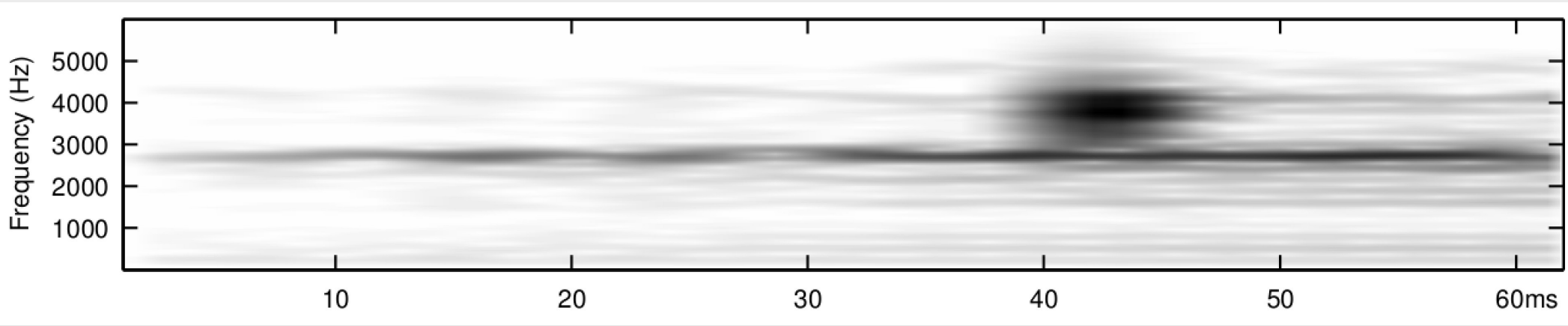
Thierry Bertin-Mahieux

University of Montreal

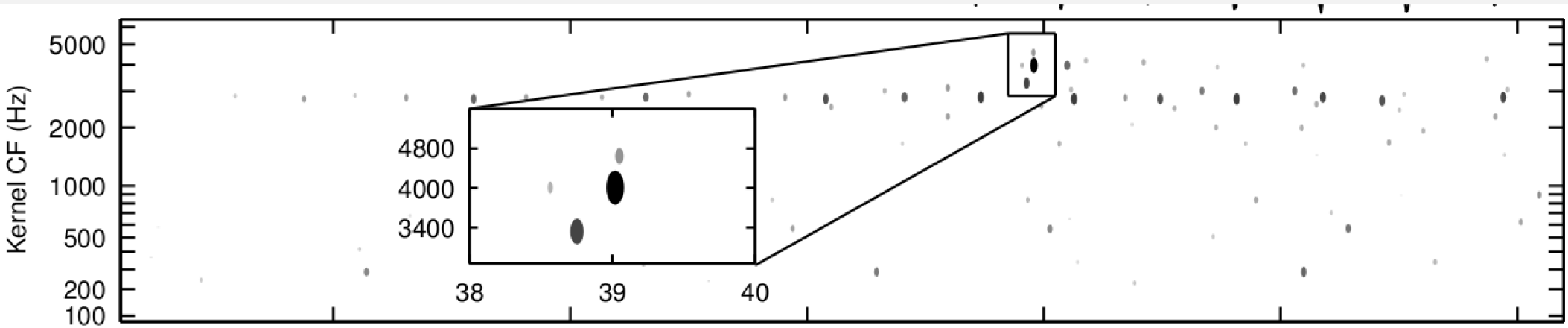
Course IFT6080 (D. Eck)

January 2008

Goal

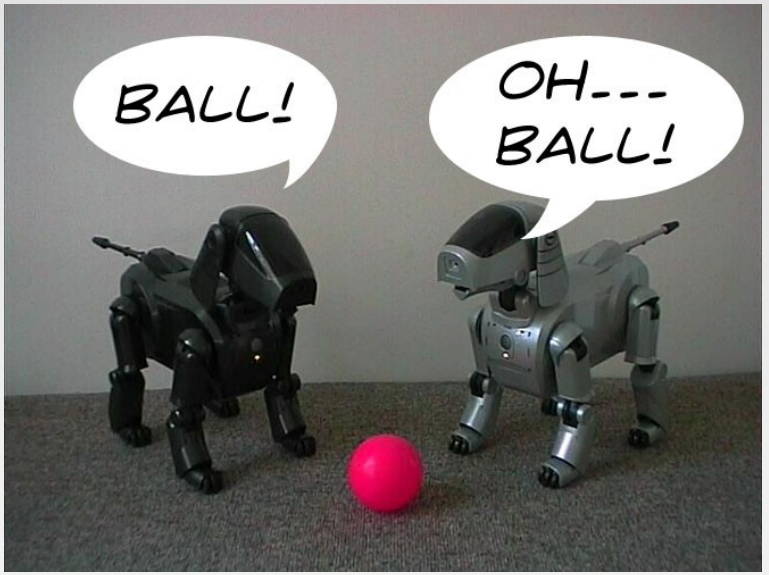


- Present articles:
  - Smith and Lewicki, Efficient auditory coding, in *Nature*, 2006
  - Smith and Lewicki, Efficient coding of time-relative structure using spikes, in *Neural Computation*, 2005



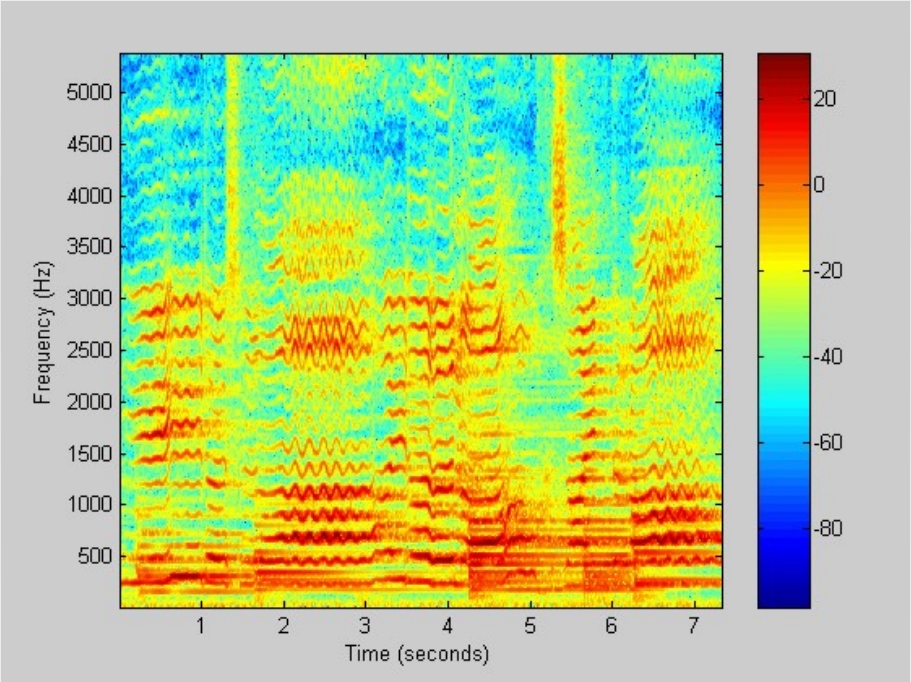
Overview

- Problems with Usual Methods
- Spike Code
- Encoding Algorithms
- Experiments
- Train Spikes
- Discussion
- More Cute Animals (if time permitting)



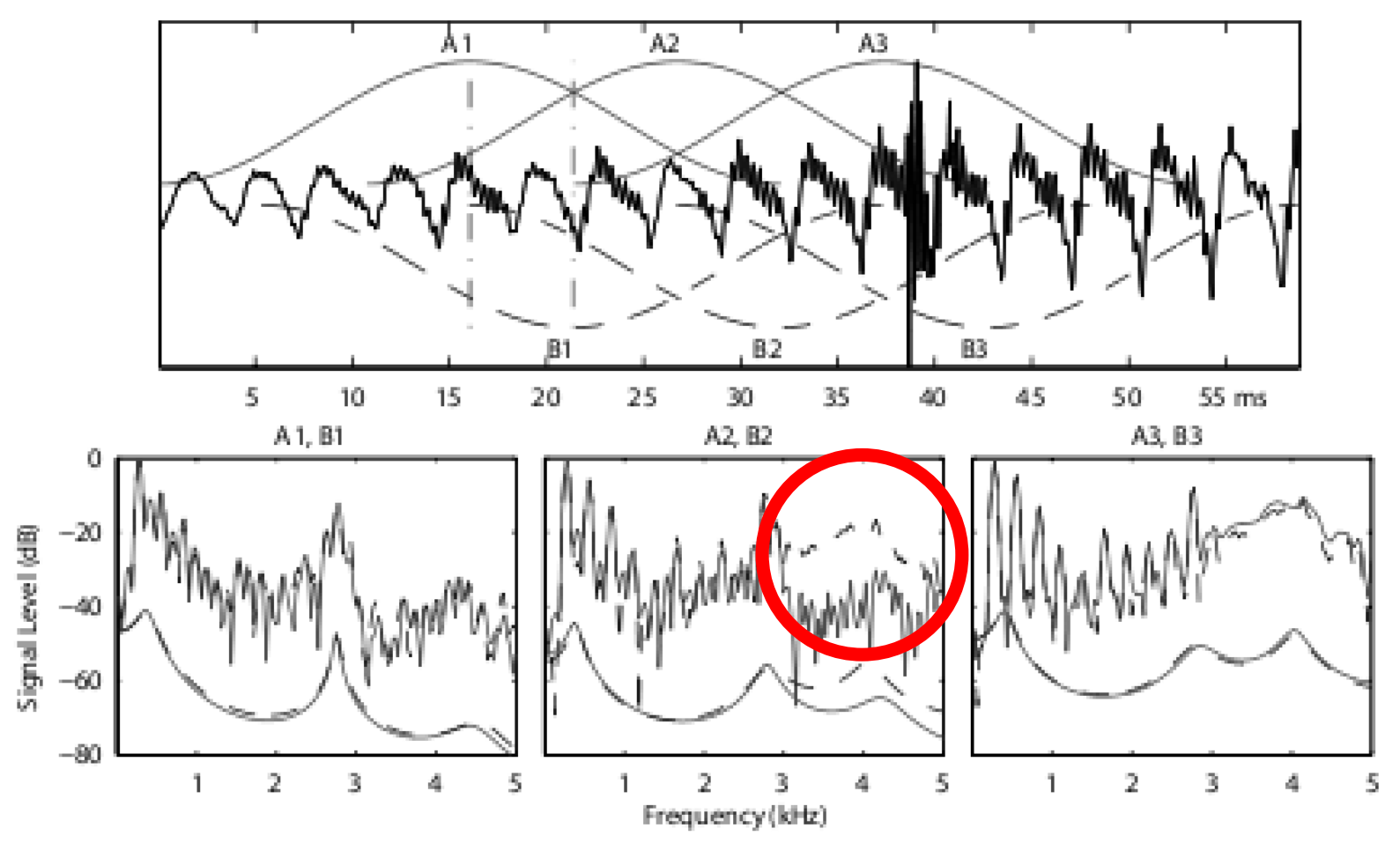
Usual Methods

- audio features, we want:
  - extract useful information
    - pitch, rythm, timbre, etc
  - dimensionality reduction
    - information in signal = 50,000 bits per second
    - intrinsec information = 200 bits per second (Rabiner & Levinson, 1981)
- Spectrogram is one possible trade-off



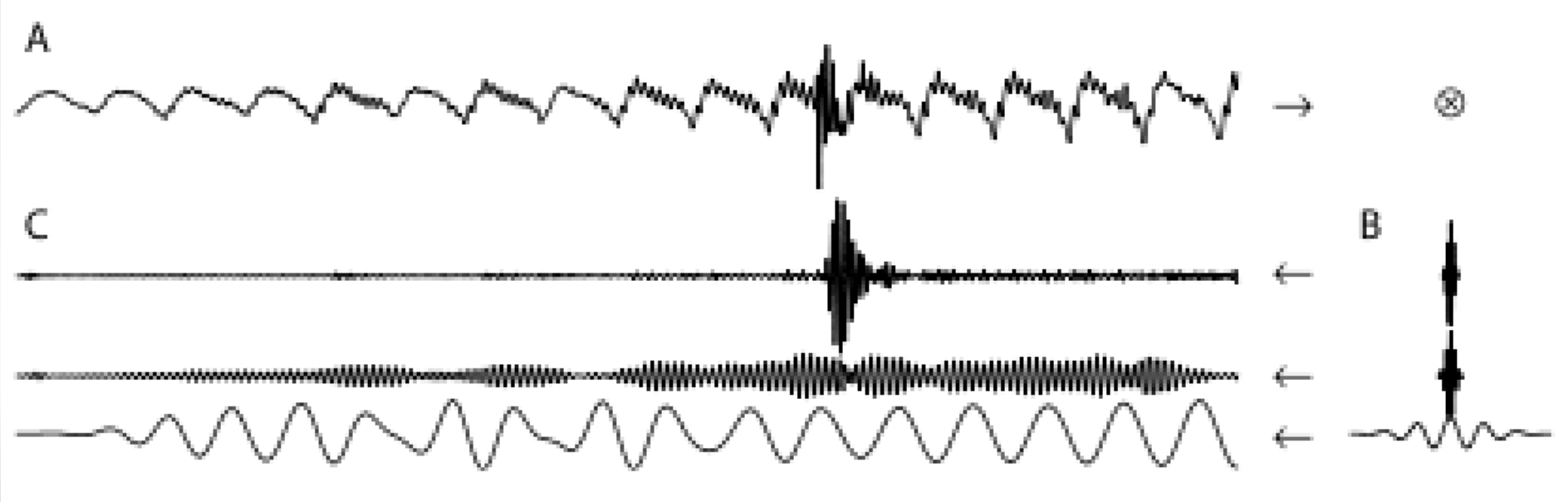
Problems with Usual Methods

- Block-Based Representations
  - sensitivity to temporal shifts



Problems with Usual Methods

- One Solution: Huge Block (Whole Signal?)
  - convolve the signal with different filters to extract meaningful information
  - no dimensionality reduction



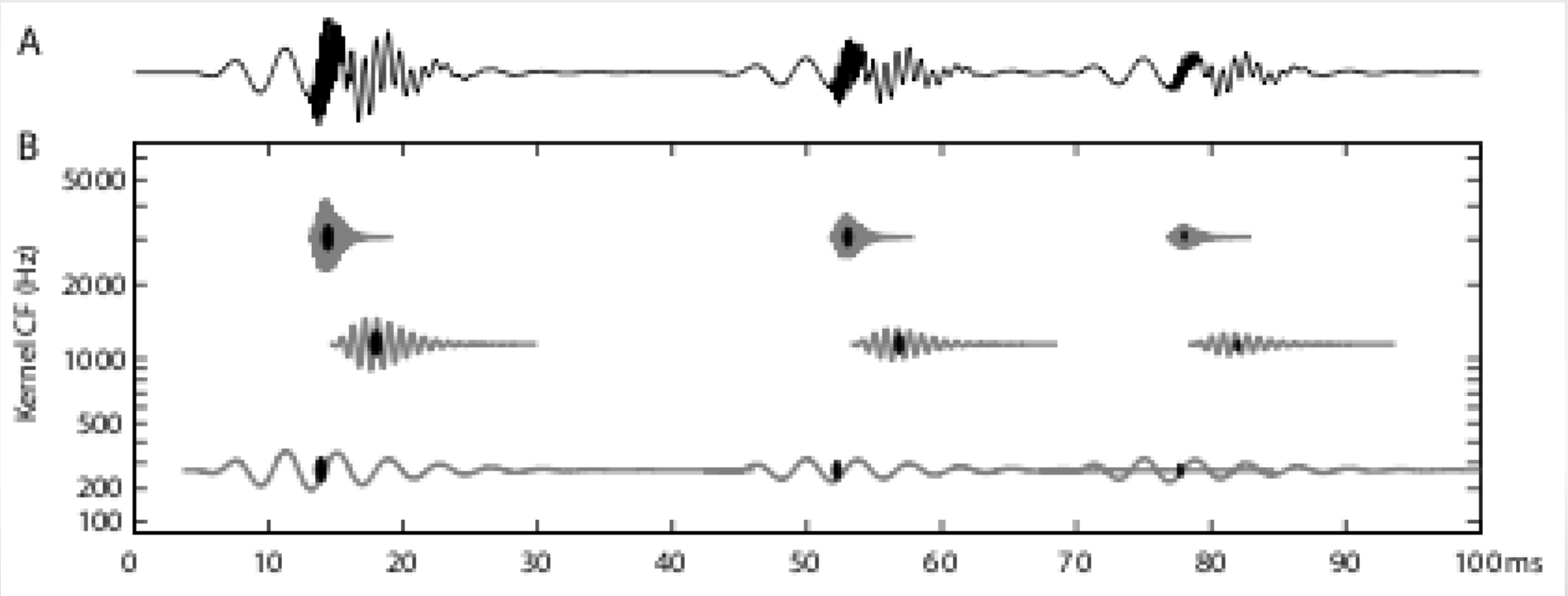
Spike Code

- Assumption
  - **sound = serie of discrete temporal events**
  - events represented by kernels
  - each kernel has a frequency and an amplitude
  - kernels fixed for the moment
- A sound can be efficiently encoded using these events

Spike Code

- Example of a Spike Code: a **Spikegram**
  - A is the signal
  - B is the encoding with 3 different kernels
  - Later on we'll only keep the centroïd

www.dramaticprairiedog.com/



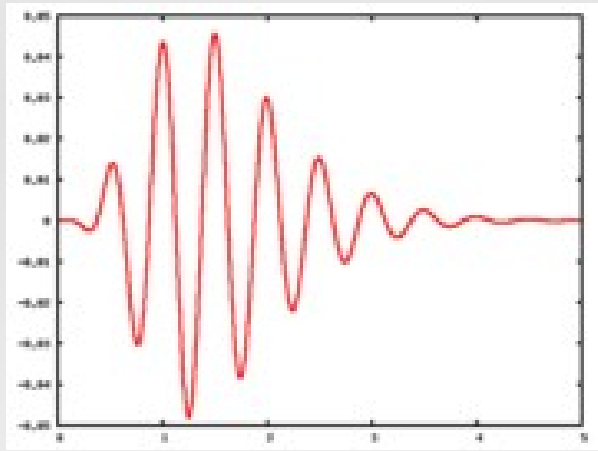
Spike Code: Maths

- Encoding with Kernels
  - Φ kernel, x(t) signal, ε noise, τ position, s coefficient
  - $x(t) = \sum_{m=1}^{M} \sum_{i=1}^{n_m} s_i^m \Phi_m(t - \tau_i^m) + \varepsilon(t)$
  - we simplify by extending kernels with zeros to the whole signal length:
  - $x(t) = \sum_{m} \int s_m(\tau) \Phi_m(t - \tau) d\tau + \varepsilon(t)$

www.curiousgeorgemovie.com

Spike Code: Filter Bank

- How to Select Appropriate Filters
  - anything works in theory!
  - special case: **gammatone**
    - $g(t) = a t^{n-1} \cos(2 \Pi f t + \phi) \mathrm{e}^{-2 \Pi b t}$
    - a amplitude
    - f frequency
    - b filter bandwith
    - biological reason: they are approximations of cochlear filters



Encoding Algorithm - 1

- Filter Threshold
  - convolve every kernel with the signal
  - keep all kernels at positions where the result energy exceeds a certain threshold
  - no correlation between kernels taken into account
  - redundancy partly compensated by a linear weighting (multiply all kernels by one value)
  - computationally fast

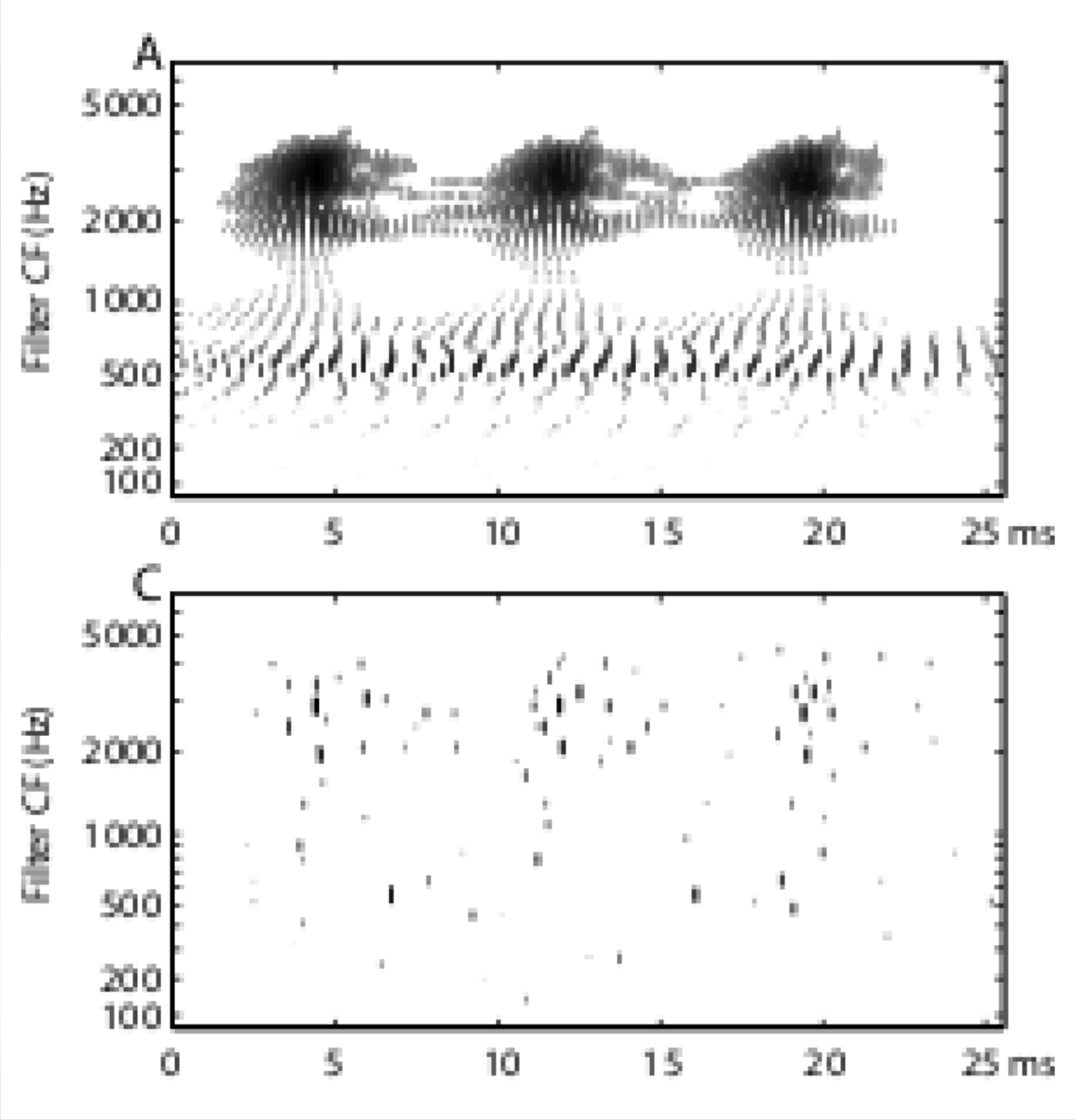
Encoding Algorithm - 2

- Matching Pursuit
  - iterative algorithm, one kernel chosen each round
  - project signal x(t) onto the kernel basis
  - $x(t)=\langle x(t)\phi_m\rangle\phi_m+R_x(t)$
  - R is called the residual, <ab> = a.b / |a|.|b|
  - each iteration we have: $R_x^{\,n}(t)=\langle R_x^{\,n}(t)\phi_m\rangle\phi_m+R_x^{\,n+1}(t)$
  - we choose $\phi_m=arg\max_m\langle R_x^{\,n}(t)\phi_m\rangle$

Encoding Algorithm - 2

- Matching Pursuit
  - we substract projection, than project residual onto kernels; new residual is orthogonal to the last kernel chosen
  - kernels span signal space => residual can decrease to any arbitrary small error value
  - computational trick: after choosing kernel Φ_n, we only look at Φ that correlates with Φ_n with a sufficent coefficient (weird trick in my mind)

Encoding Algo. - Comparison 1



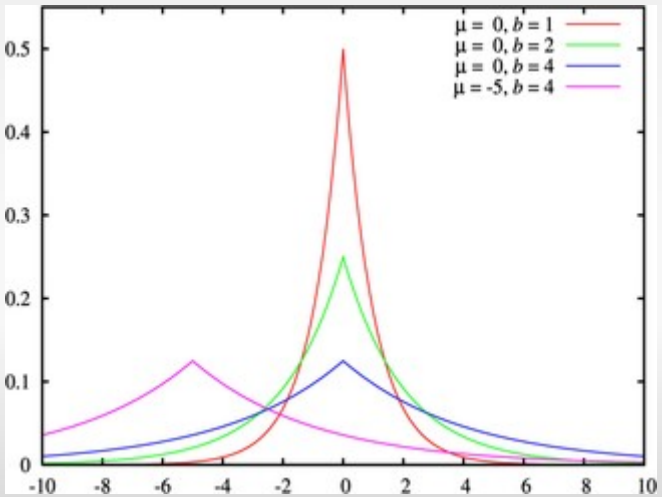
- A - Filter threshold
- C - Matching pursuit
- Second algo is much sparser on the same signal
- See signal-to-noise ratio in a few slides to compare the quality of the encodings

Encoding Algo. - Optimization

- MAP (maximum a posteriori)
  - rewrite signal: $x = As + \epsilon$
  - $A = [C(\phi_1) \ldots C(\phi_m)]$
  - $C(\Phi_i)$ contains all time shifts of kernel Φ_i
  - to find the best τ_i^m and s_i^m we look at:
  - $\hat{s} = arg\max_s P(s|x, A) = arg\max_s P(x|A, s)P(s)$
  - we assume gaussian noise and P(s) very sparse

Encoding Algo. - Optimization

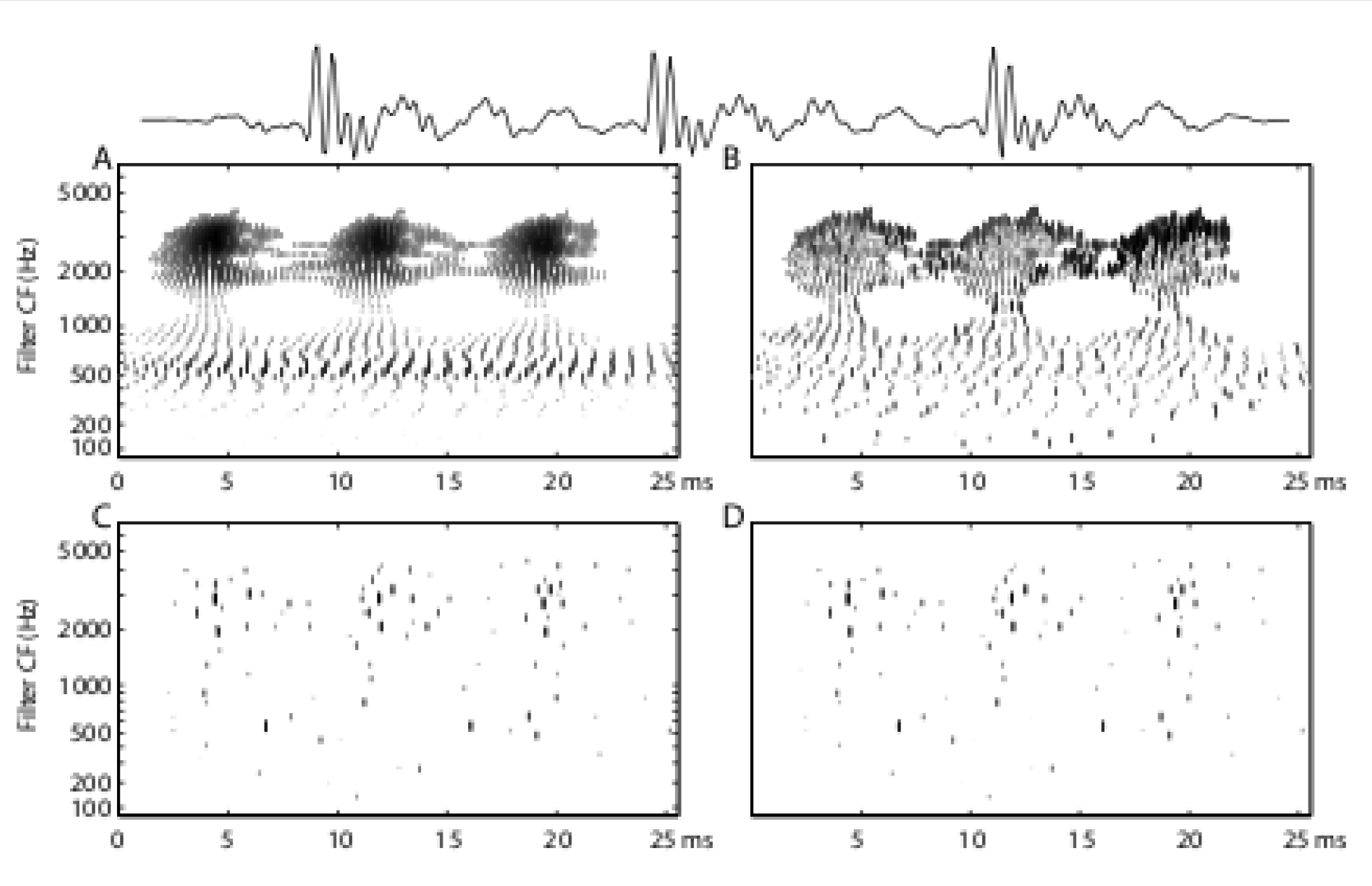
- MAP (maximum a posteriori)
  - we can optimize by gradient ascent:
  - $\frac{\partial}{\partial s}\log P(s|A,x) \propto A^T(x - A s) + z(s)$
  - $z(s) = (\log p(s))'$
  - z(s) assumed to follow a Laplacian, but can follow something else
  - reminder: $\frac{\partial x}{\partial z} = \frac{\partial x}{\partial y}\frac{\partial y}{\partial z}$
  - Laplacian:



Encoding Algo. - Optimization

- MAP (maximum a posteriori)
  - can be use on its own using all possible kernels
    - computationally expensive!!!
  - instead, used as a second training step after filter threshold matching pursuit
  - we apply MAP only to the preselected spikes

Encoding Algo. - Comparison 2

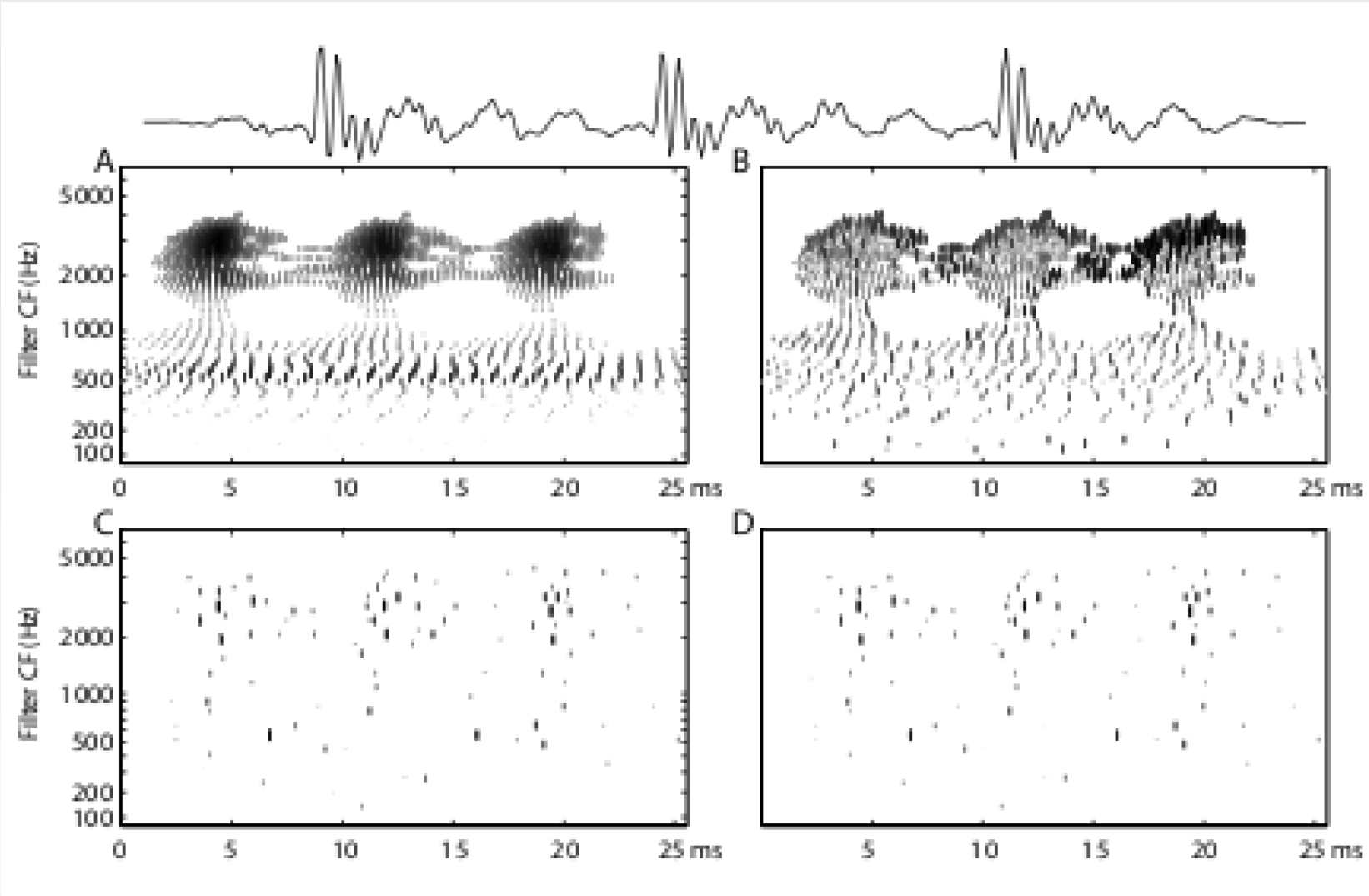


- Algo 1 and 2 (A and C), then optimized (B and D)

Signal-to-Noise Ratio

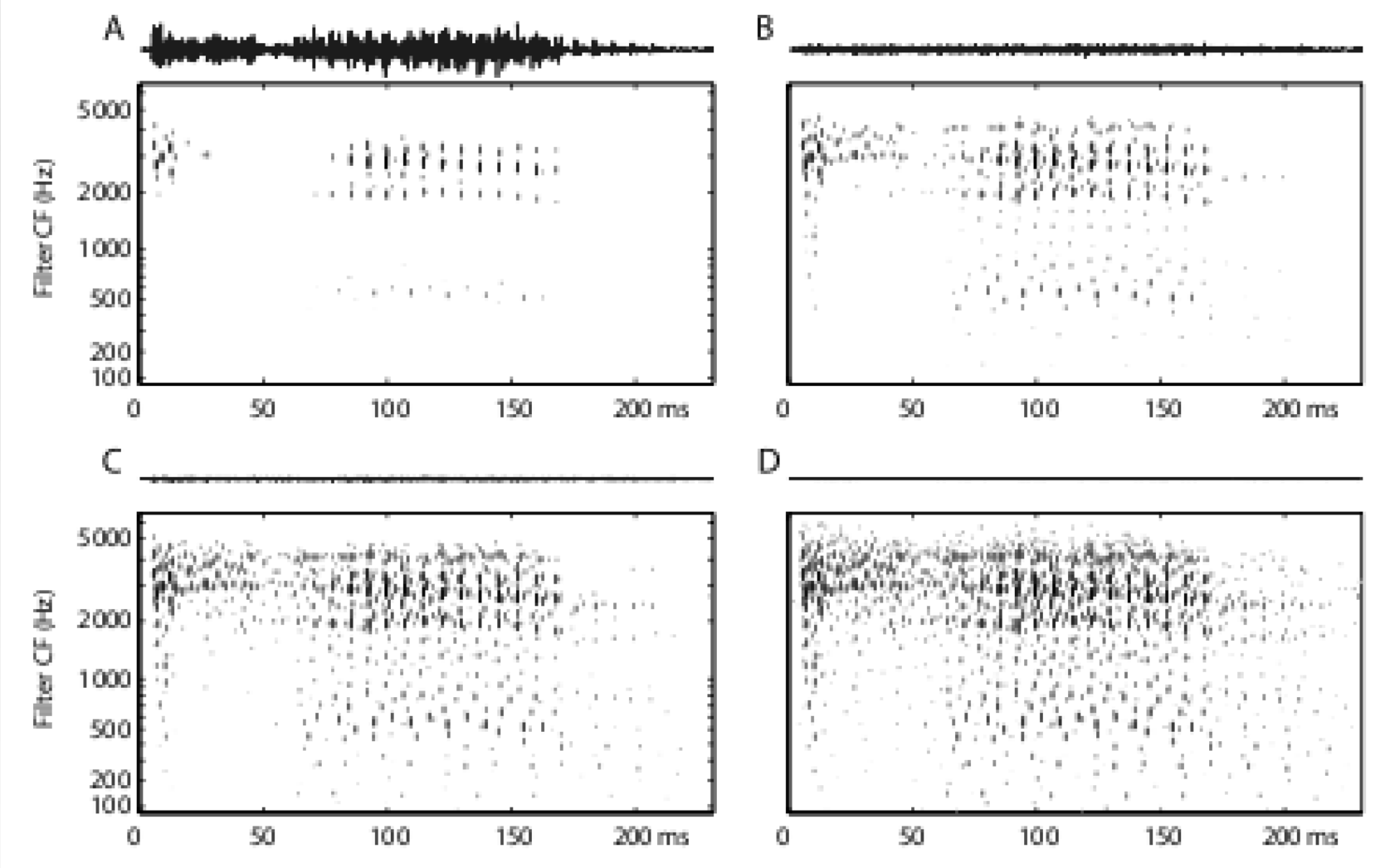
- How to Compare Encodings? SNR
  - reconstruct signal, compute residual error ε
  - power of a signal f(t): $P_f = \lim_{t \to \infty} \frac{1}{T} \int_{\frac{-T}{2}}^{\frac{T}{2}} (|f(t)|)^2 dt$
  - we compute P_ε and P_o, power of residual error and power of original signal
  - $SNR = 10 \log_{10}(P_o / P_\epsilon)$
  - so, high SNR is good!

Encoding Algo. - Comparison 3



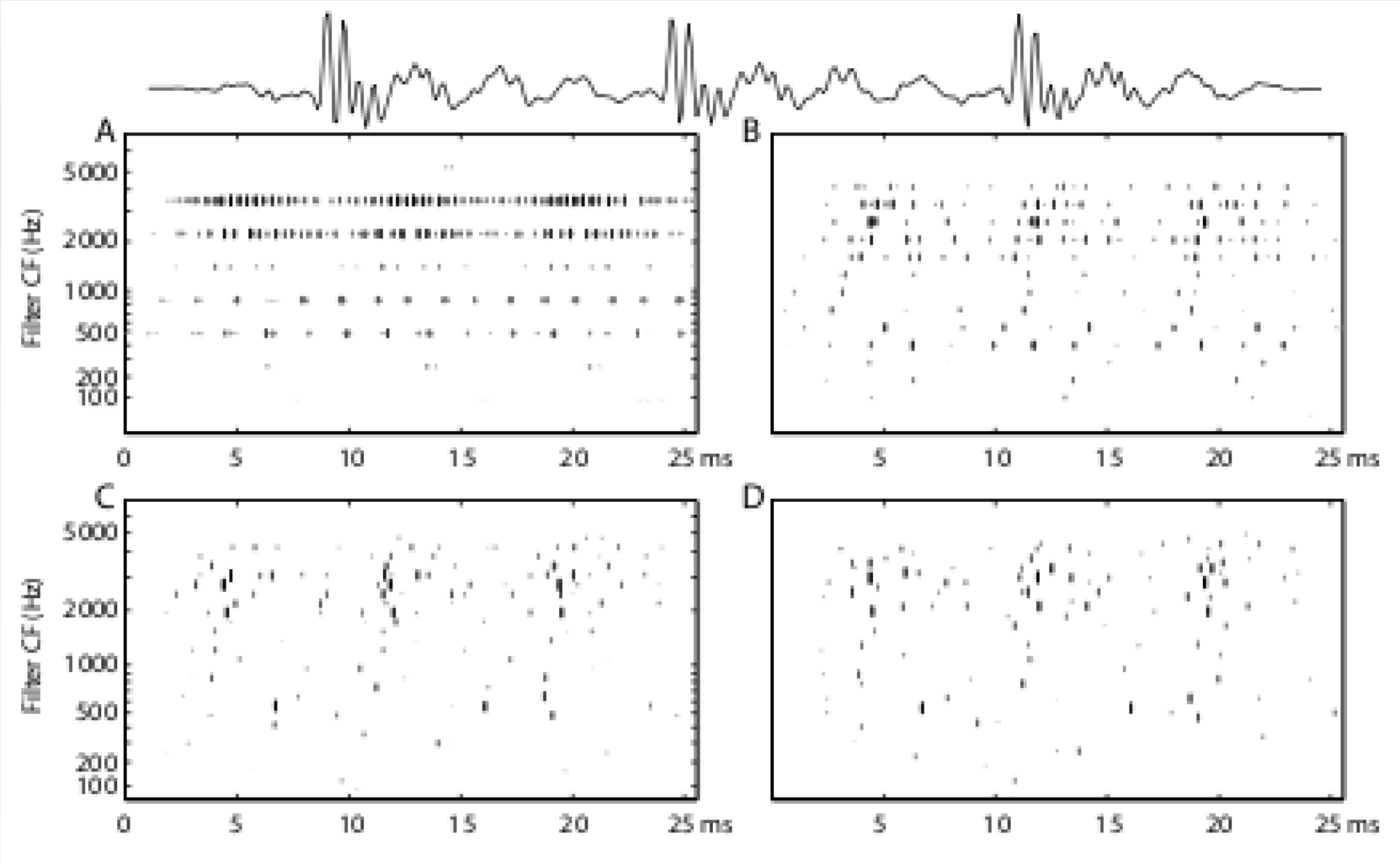
- **SNR** (db)
- A – 18.7
- **B – 90.1**
- **C – 30.4**
- D – 33.0

Parameters – Number of Spikes



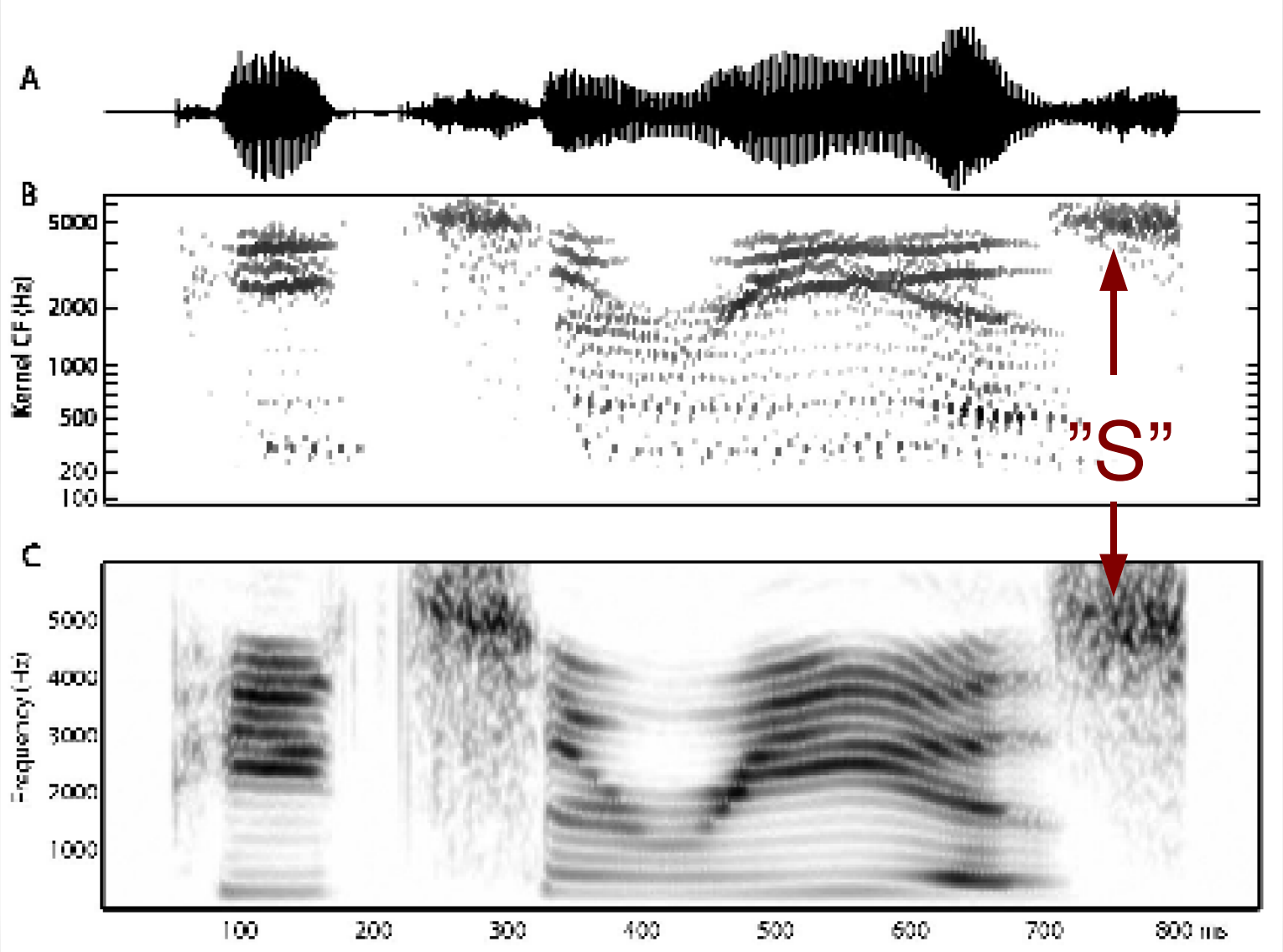
- Matching pursuit
- Num. of spikes: 400, 1600, 3100 and 5500 spikes per sec.
- DBs are 10, 20, 30 and 40

Parameters – Number of Kernels



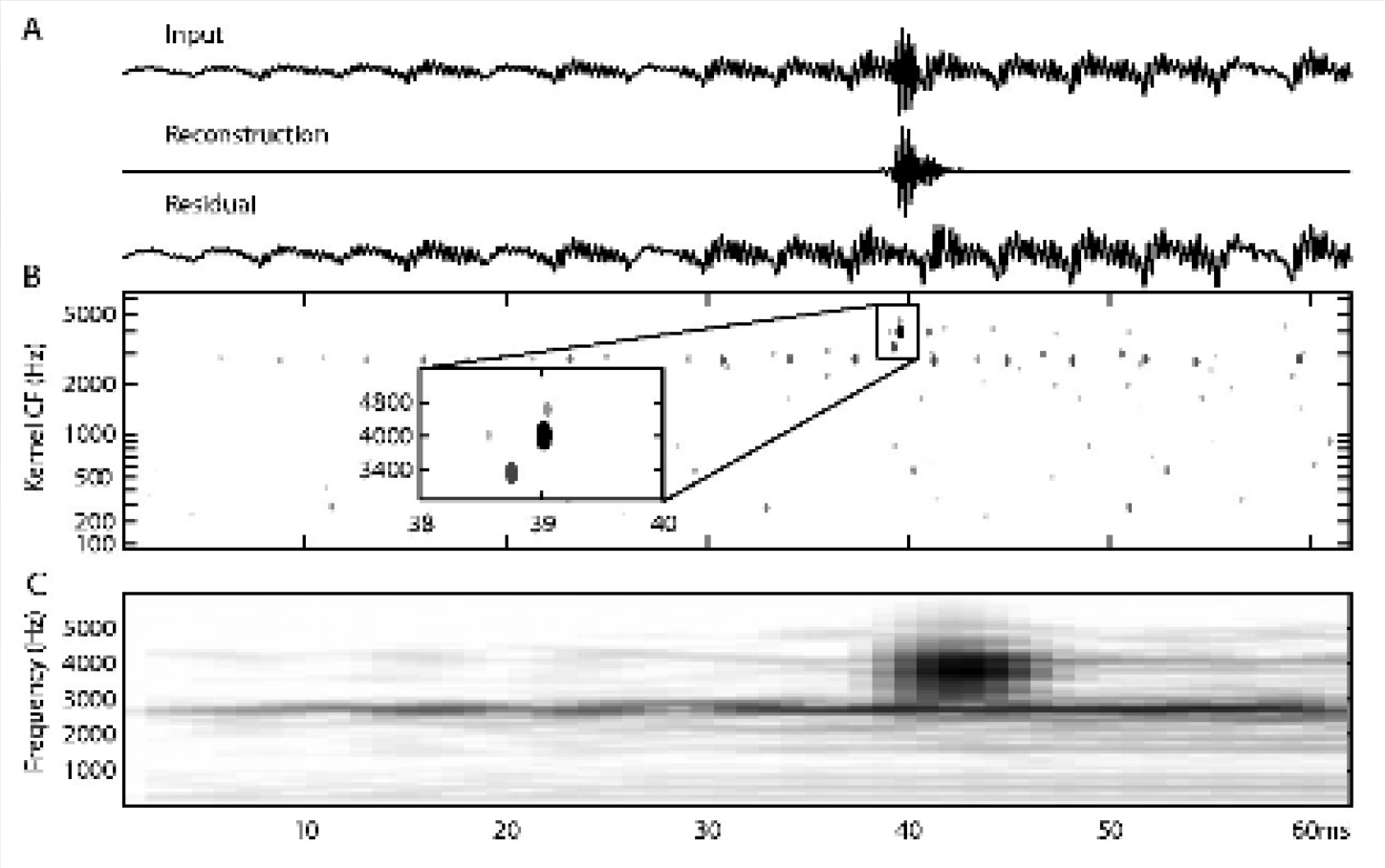
- Matching pursuit
- More kernels is better
- Here: 8, 16, 32, 64
- Number of spikes: 12011, 1167, 497 and 479

Comparison with FFT



- encoding of the word "pizzerias"
- both reveals the large-scale structure (like "s")
- timing information like phase more precise with spikegram

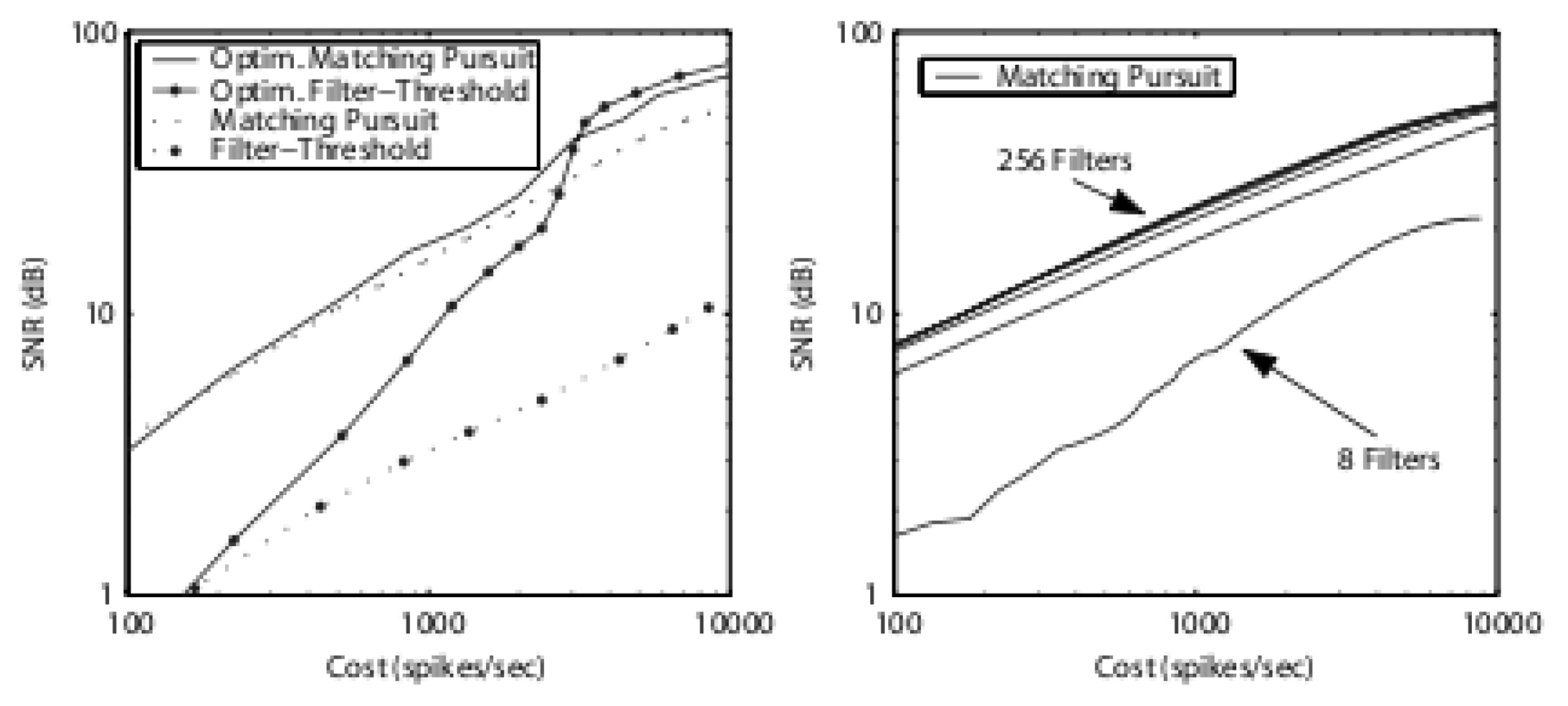
Comparison with FFT - 2



- spikegram to represent the ”t” in ”vietnamese”, a transient
- spikes of the ”t” are isolated (see reconstruction in A)

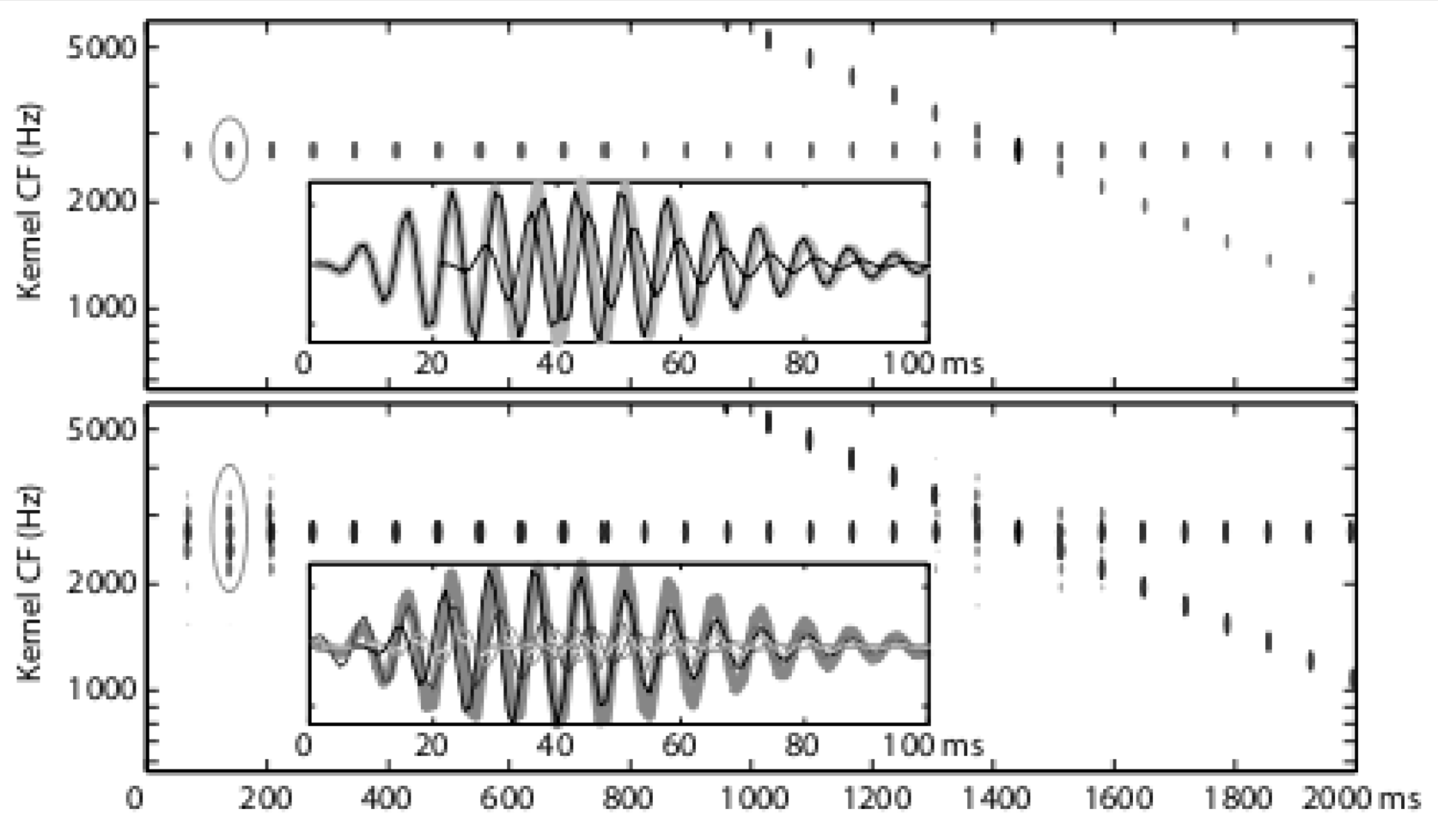
Wikipedia: a transient is a short-duration signal that represents a non-harmonic attack phase of a musical sound or spoken word. It contains a high degree of non-periodic components and a higher magnitude of high frequencies than the harmonic content of that sound. Transients do not directly depend on the frequency of the tone they initiate.

Code Efficiency



- Number of spikes per second and corresponding DB. Matching pursuit best trade-off. Bigger filter bank implies better encoding.
- If you use a lot of spike, optimized filter threshold beats matching pursuit (see next slide)

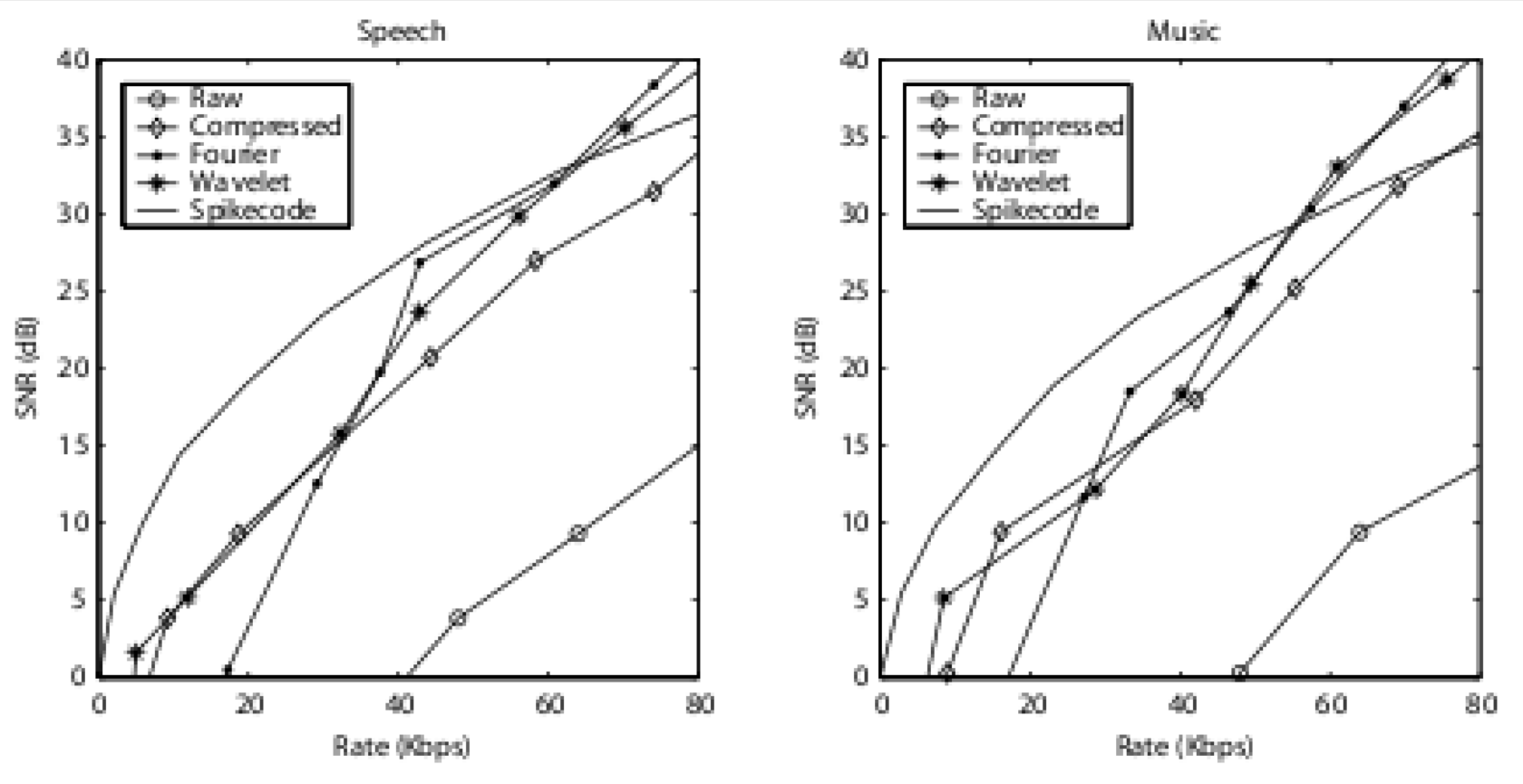
Matching Pursuit Flaw



- Original sound produded by 2 gamma-tones seperated by 2 msec
- MAP, then matching pursuit

- Original sound is thick gray curve, found kernels are thin lines.
- Matching pursuit can't find the good filters because the first one is innacurate, then only tries to compensate.

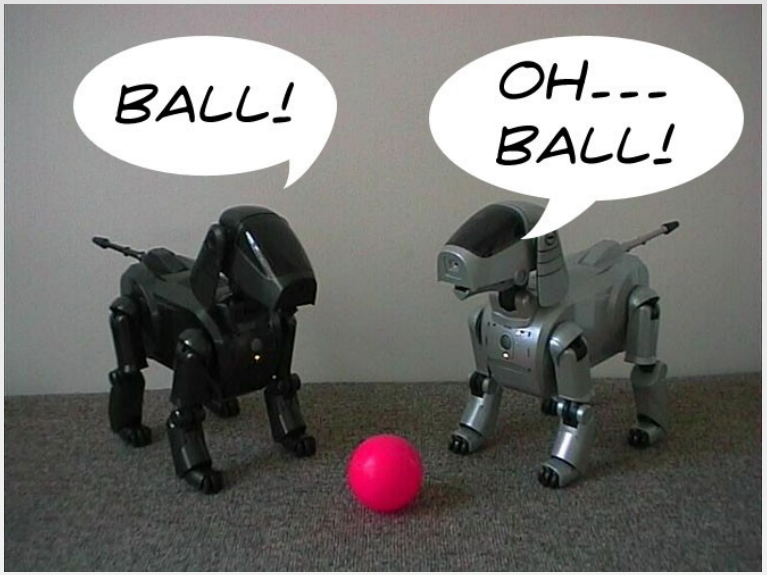
Code Efficiency 2



- SNR in DB function of encoding rate using different encoding
- Spikecode good!

Overview

- Problems with Usual Methods
- Spike Code
- Encoding Algorithm
- Experiments
- Train Spikes
- Discussion
- More Cute Animals (if time permitting)



Learning Spikes

- Gammatones are fine, but can we fine-tune them for a precise task?
  - motivation: our auditory system is probably adapted to ”useful” sounds, like speech
- 3 kinds of sounds used here
  - mammalian vocalizations
  - ambient sounds (wind, flowing water, ...)
  - transients (snapping twigs, crunching leaves, ...)

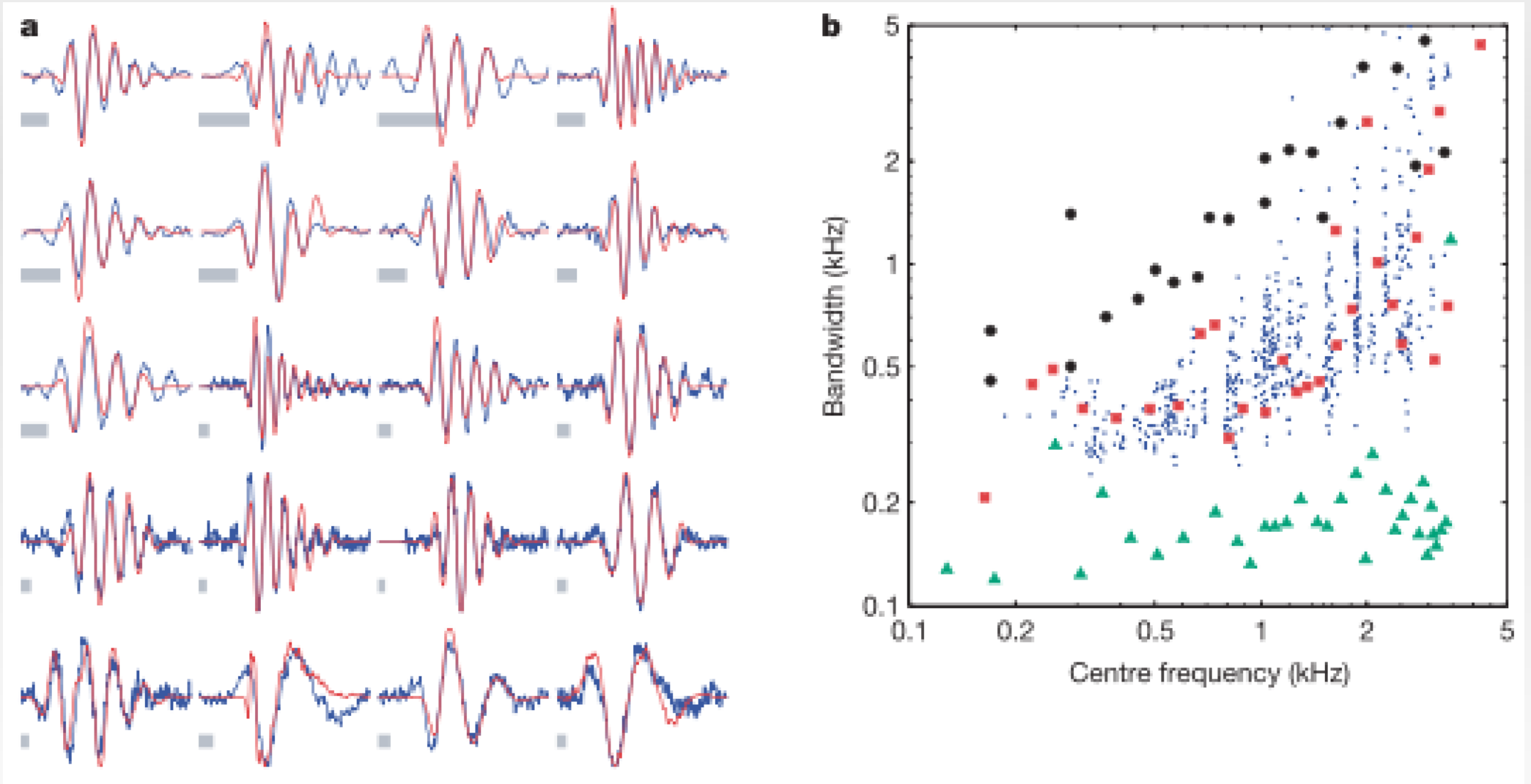
Learn Spikes – Maths

- Signal function of spikes :
- $p(x|\phi)=\int p(x|\phi,s)p(s)ds\approx p(x|\phi,\hat{s})p(\hat{s})$
- $\hat{s}$ approximation of the posterior maximum, comes from the set of coeff. from matching purs.
- noise in the likelihood assume to be gaussian, and p(s) sparse
- we'll optimize by gradient ascent p(x|Φ), or for conveniance log(p(x|Φ))

Learn Spikes – Maths 2

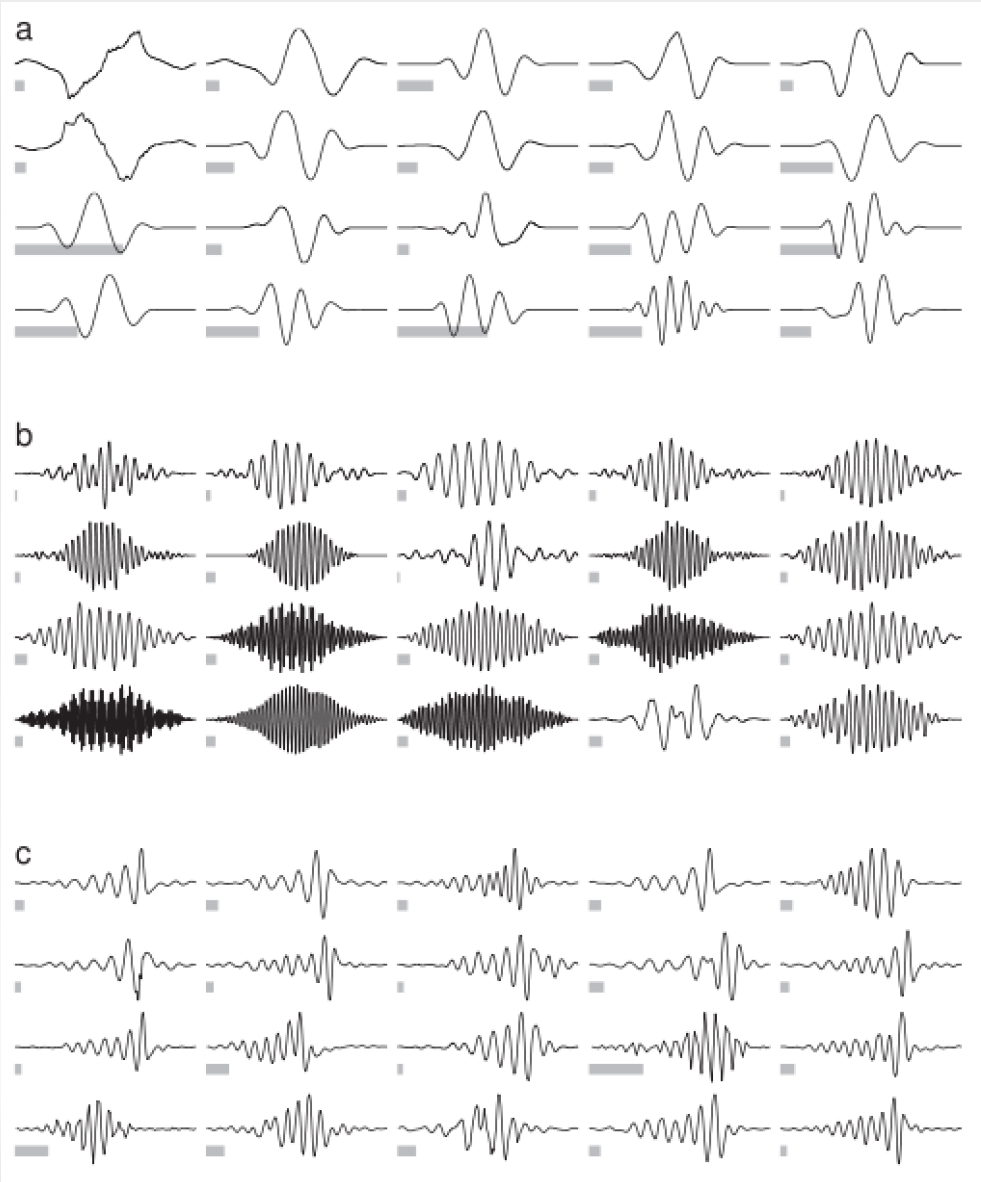
- $$\frac{\partial}{\partial \phi_m} \log(p(x|\phi)) = \frac{\partial}{\partial \phi_m} [\log(p(x|\phi, \hat{s})) + \log(p(\hat{s}))]$$
- $$\frac{\partial}{\partial \phi_m} \log(p(x|\phi)) = \frac{-1}{2\sigma_\epsilon^2} \frac{\partial}{\partial \phi_m} \| x - \sum_{m=1}^{M} \sum_{i=1}^{n_m} \hat{s}_i^m \phi_m (t - \tau_i^m) \|^2$$
- $$\frac{\partial}{\partial \phi_m} \log(p(x|\phi)) = \frac{1}{\sigma_\epsilon^2} \sum_i \hat{s}_i^m [x - \hat{x}]_{\tau_i^m}$$
- reminder: $\frac{\partial x}{\partial z} = \frac{\partial x}{\partial y} \frac{\partial y}{\partial z}$
- $[x - \hat{x}]_{\tau_m^i}$ is the residual error of kernel τ_i^m at ϕ_m position
- we update s, τ and Φ, kernels can get longer or shorter depending on the frequency

Learn Spikes – Experiments



- (A) learned gammatones (red) fit revcor functions (blue) from the cat auditory nerve fibres. (B) blue cat, red learned, black learned only on environmental sounds, green only on animals vocalizations

Learn Spikes – Experiments 2



- learned kernels for different training sets:
  - a – environmental sounds alone
  - b – vocalizations
  - c – reversed speech

Discussion

- Computational cost of all this?
- Are these methods good for noise reduction? speaker separation?
  - what are the limits of the additive framework of independant audio events
- More kernels, task specific kernels?
  - ISMIR08 -> learn genre specific kernels for classification...

Questions?

References

- M. Lewicki and T. Sejnowski, Coding time-varying signals using sparse, shift-invariant representations, in *NIPS*, 1998
- L. Rabiner and S. Levinson, Isolated and connected word recognition: theory and selected applications, in *IEEE Trans. Communications,* 1981
- E. Smith and M. Lewicki, Efficient auditory coding, in *Nature*, 2006
- E. Smith and M. Lewicki, Efficient coding of time-relative structure using spikes, in *Neural Computation*, 2005