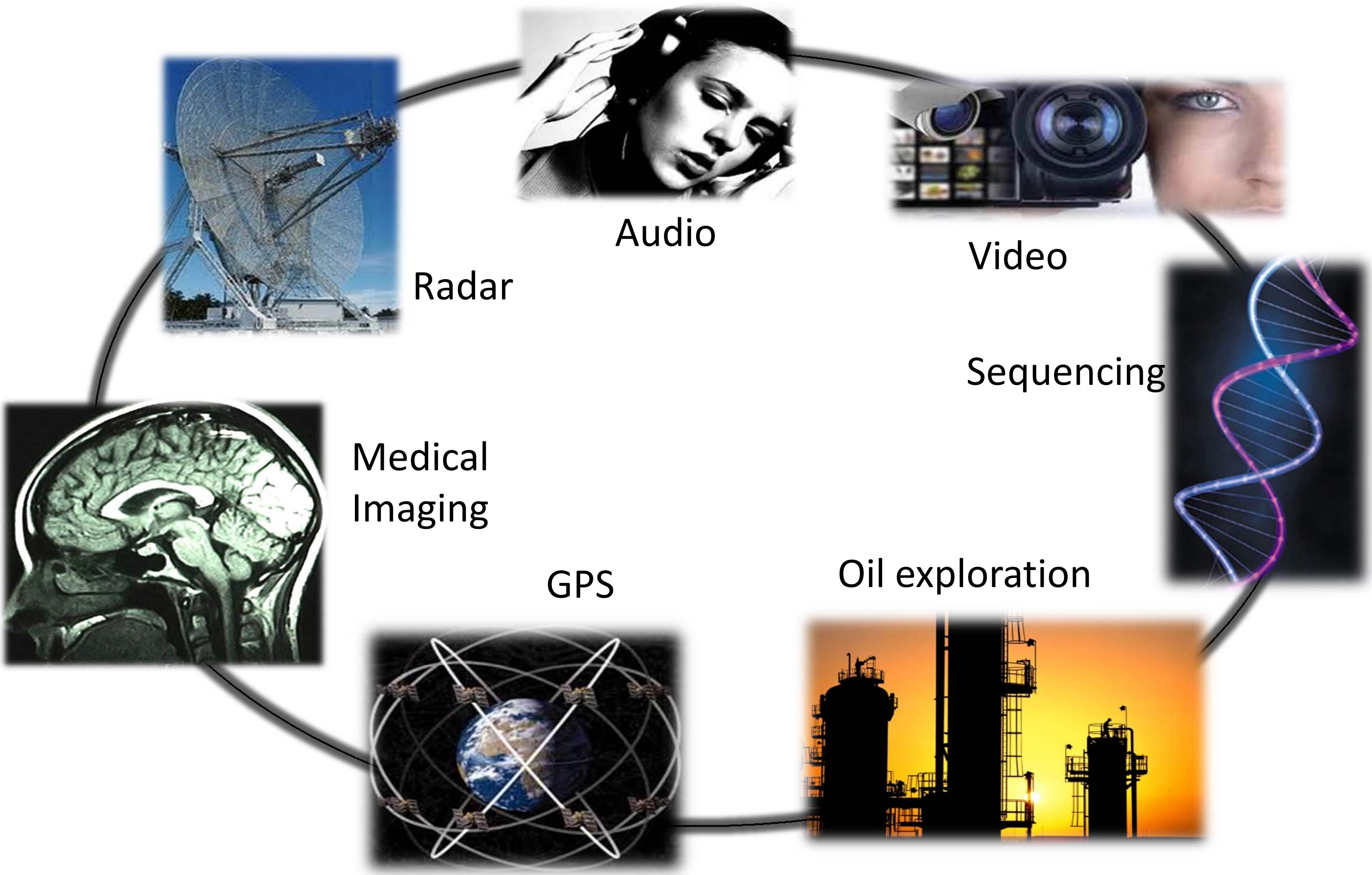
The Sparse Fourier Transform

Haitham Hassanieh

Piotr Indyk   Dina Katabi   Eric Price

Fourier Transform Is Used Everywhere



Computing the Discrete Fourier Transform

- Naïve Algorithm $O(n^2)$

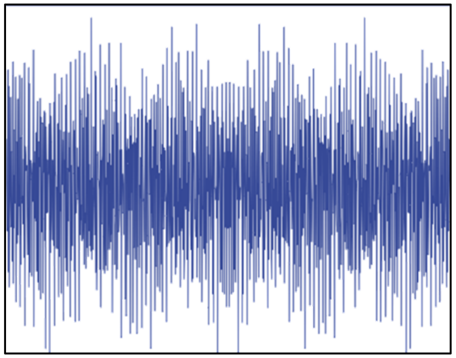
$$\hat{x}_f = \mathbf{F}\, x_t$$

- In 1965, Cooley and Tukey introduced the FFT which computes the frequencies in $O(n \log n)$
- But … FFT is too slow for BIG Data problems

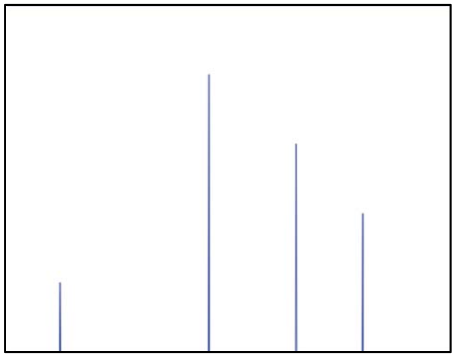
Can we design a sublinear Fourier algorithm?

Idea: Leverage Sparsity

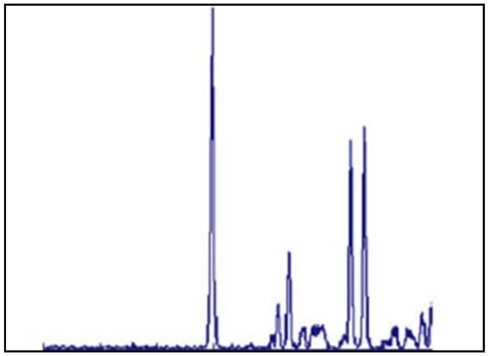
Often the Fourier Transform is dominated by a few peaks



Time Signal



Sparse Freqs.



Approximately Sparse Freqs.

Sparse FFT computes the DFT in sublinear time

Sparsity appears in video, audio, seismic data, telescope/satellite data, medical tests, genomics

Benefits of Sparse FFT

- Faster computation → Scalable to larger datasets

- Use only samples of the data
  - → Lower acquisition time
  - → Less communication bandwidth

- Lower power consumption

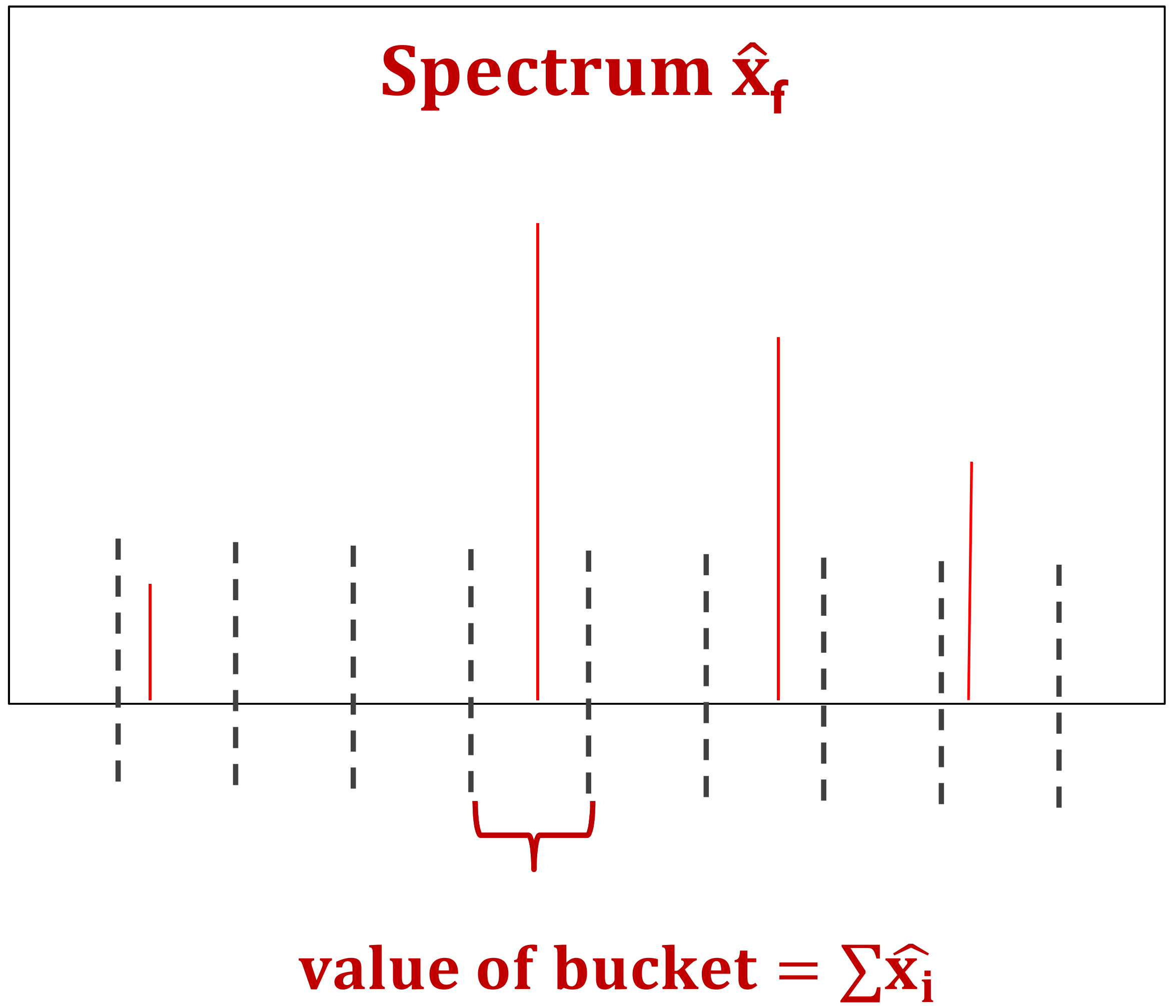
How Does Sparse FFT Work?

1- Bucketize

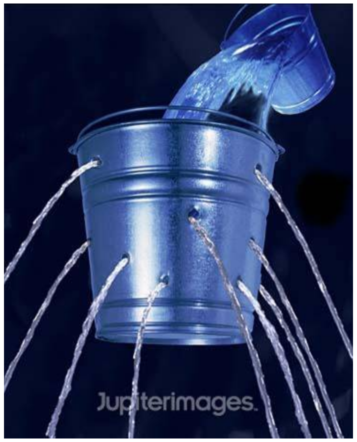
Divide spectrum into a few buckets

2- Estimate

Estimate the large coefficient of the non-empty buckets

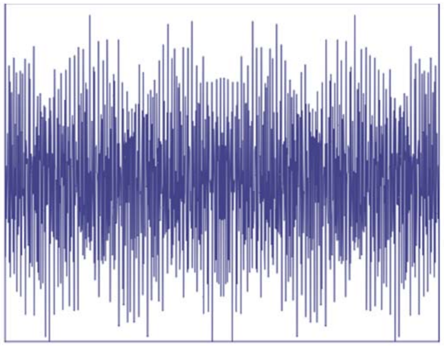


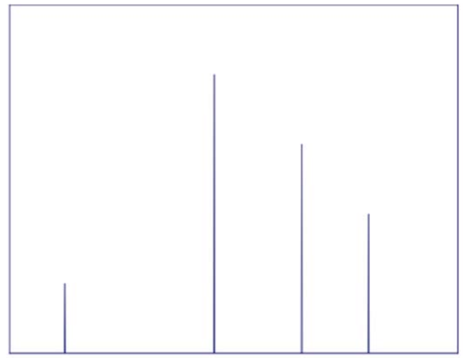
Rules of the Game



- Fast bucketization in sublinear time
- Avoid leaky buckets
- Which is the big frequency in a bucket?
- Deal with collisions

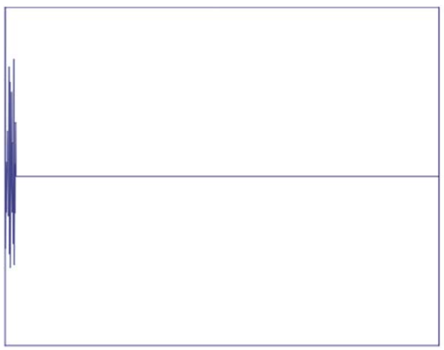
Fast Bucketization

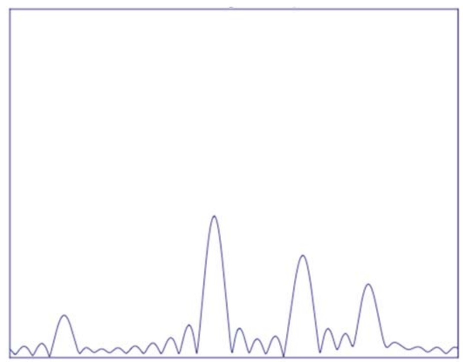

Time Domain Signal


Frequency Domain

n-point DFT : $n\log(n)$

$\mathbf{x} \rightarrow \hat{\mathbf{x}}$

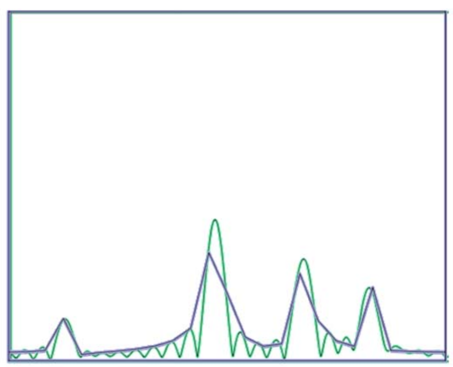

Cut off Time signal


Frequency Domain

n-point DFT: $n\log(n)$
using first B samples

$\mathbf{x} \times$ **Boxcar** $\rightarrow \hat{\mathbf{x}} *$ **sinc**


First B samples


Frequency Domain

B-point DFT of first
B terms: $B\log(B)$

Alias $(\mathbf{x} \times$ **Boxcar**$)$

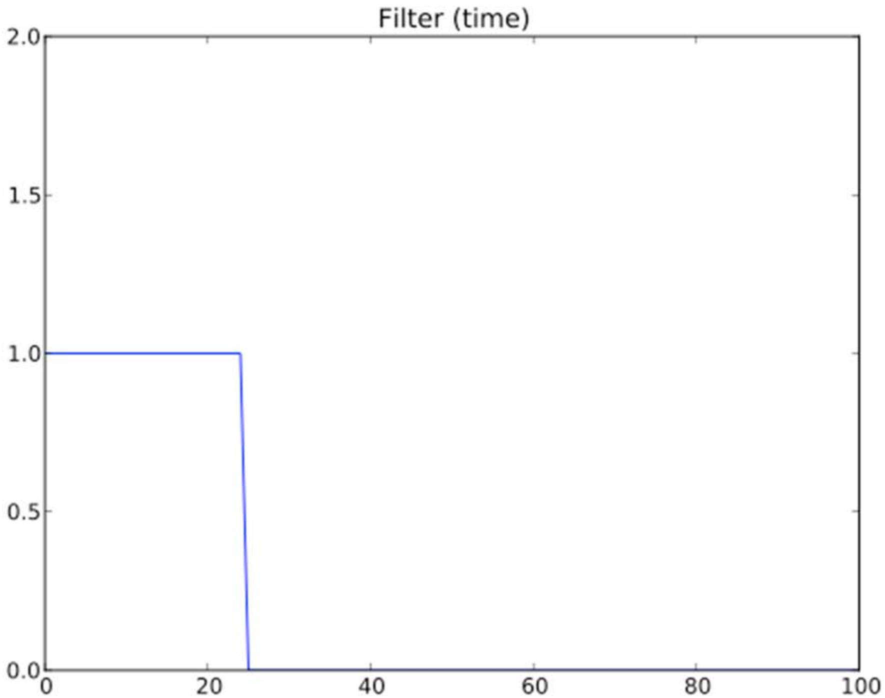
$\downarrow$

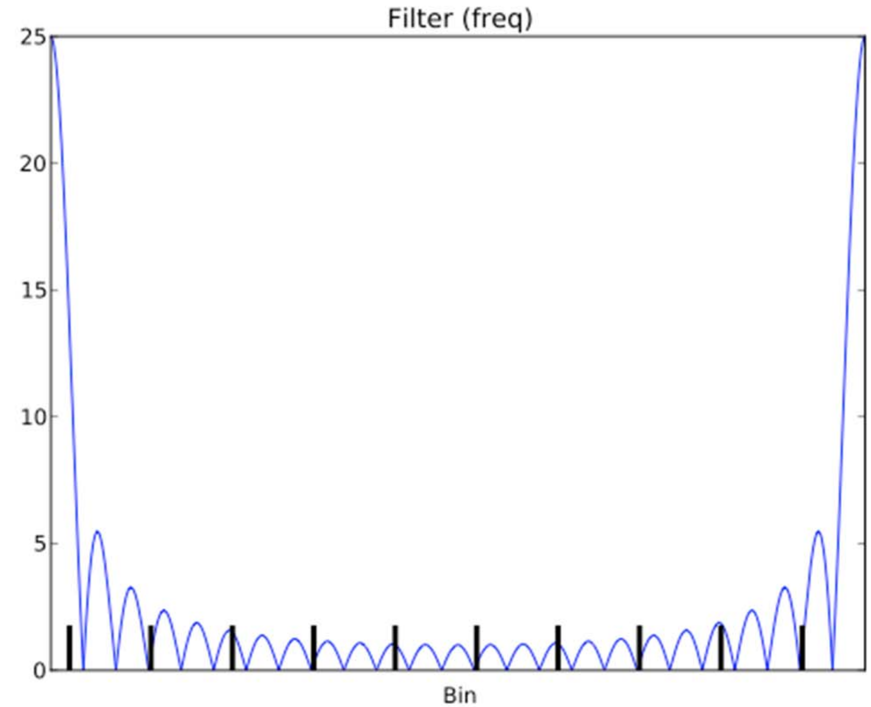
Subsample $(\hat{\mathbf{x}} *$ **sinc**$)$

But these are leaky buckets

- Leakage
  - value of bucket = **Subsample** $(\hat{x} * \textbf{sinc})$
  - sum over all frequencies weighted by sinc
- Solution
  - Replace **sinc** with a better **Filter**
  - **GOAL : Subsample** $(\hat{x} * \textbf{Filter})$ = sum of the frequencies that hash to the bucket
- Which Filter satisfies the above?

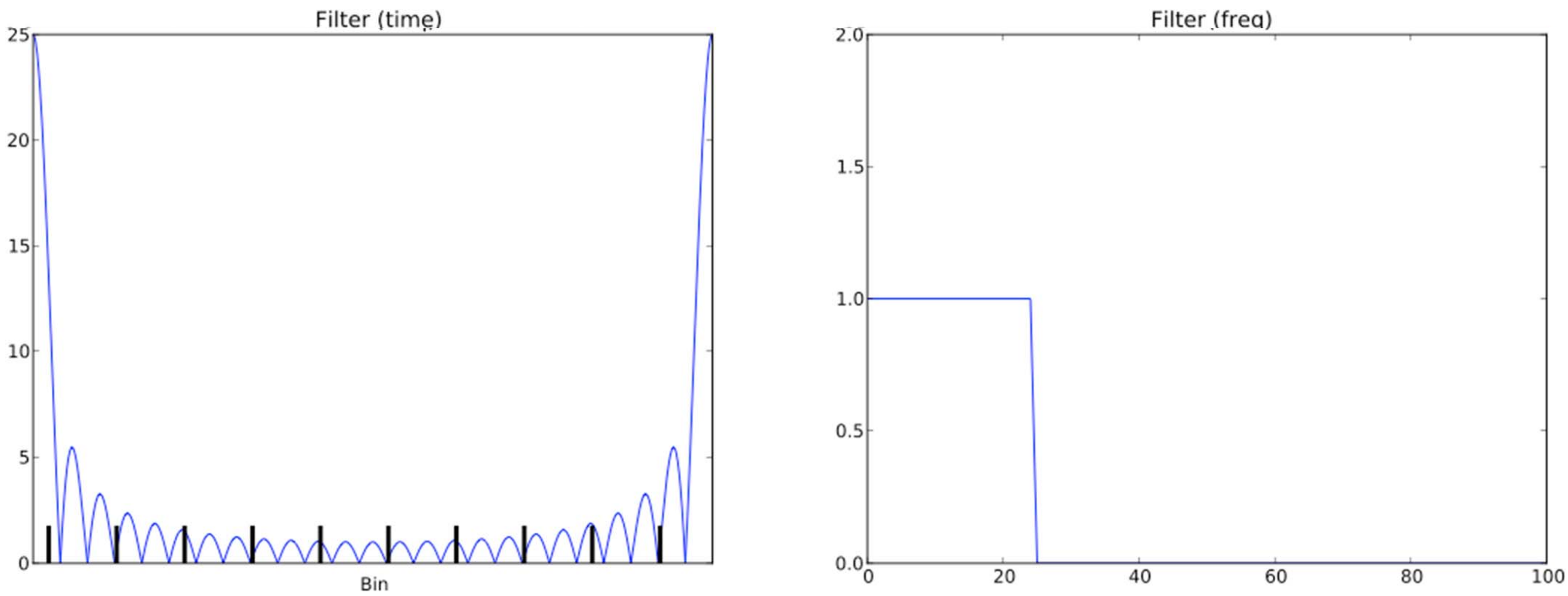
Filters: Boxcar (in the time domain)





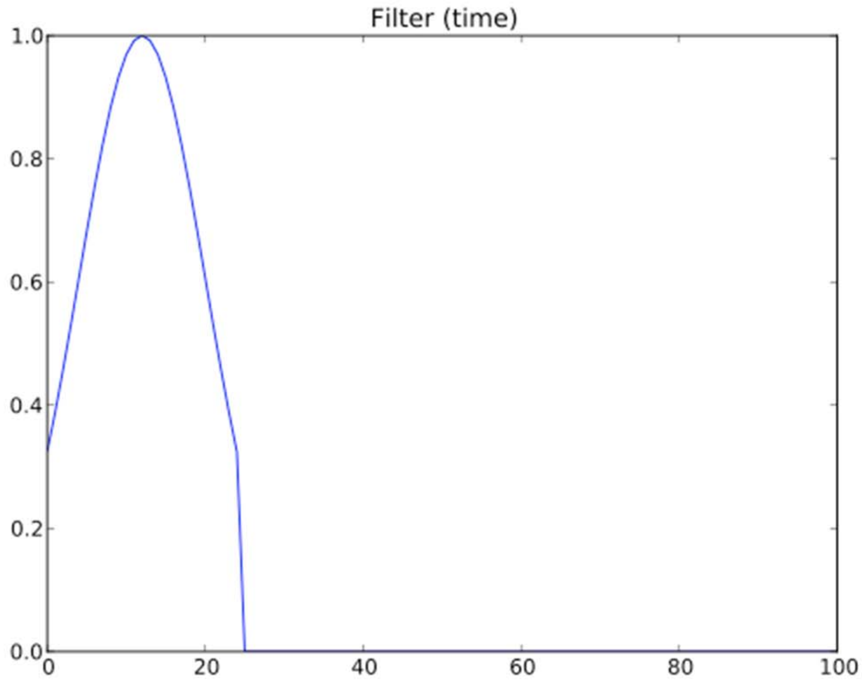
- Boxcar → Sinc
  - Polynomial decay
  - Leaking many buckets

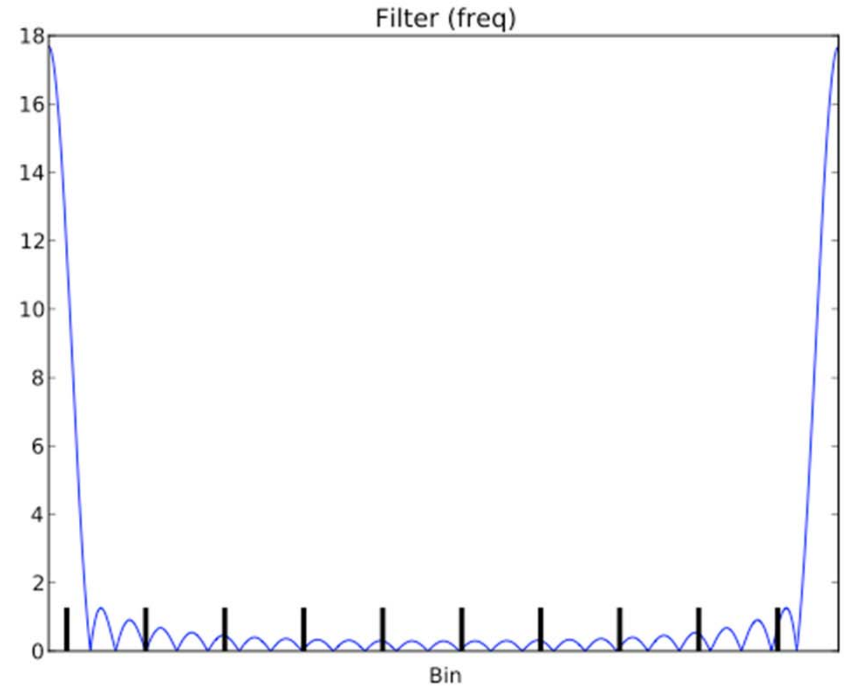
Filters: Sinc (in the time domain)



- Sinc → Boxcar
  - Large time domain support
    → linear time complexity

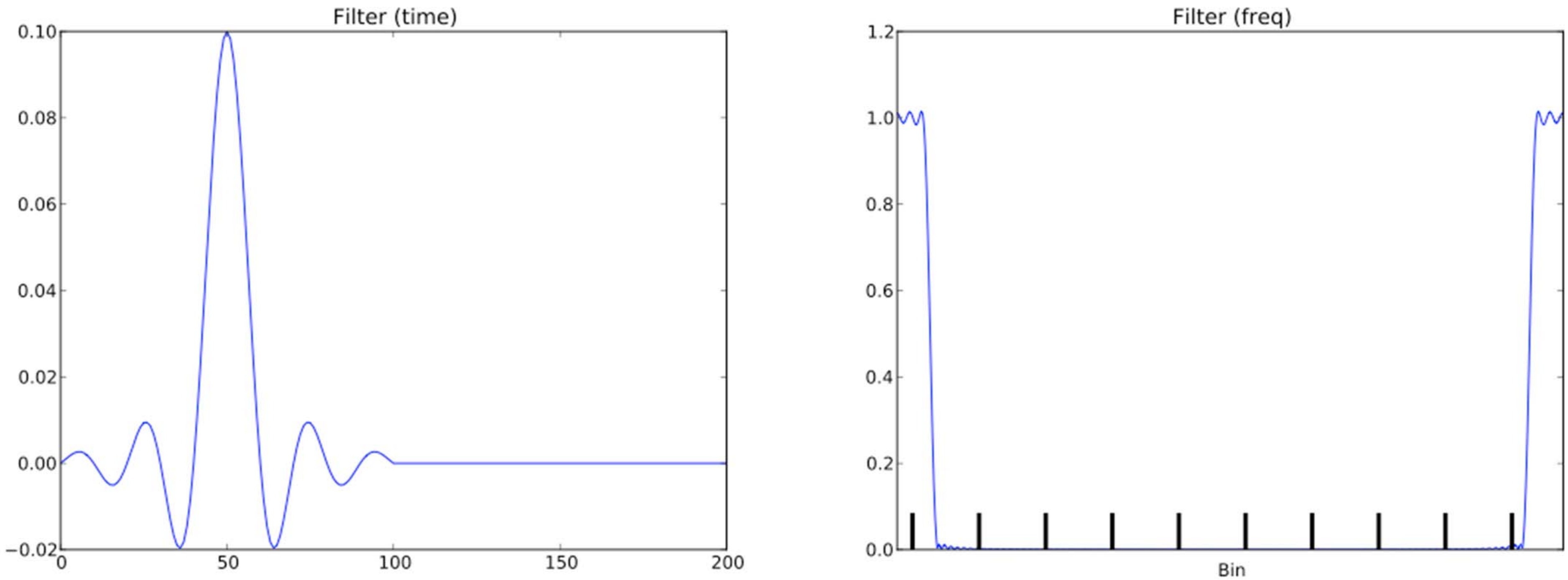
Filters: Gaussian (in the time domain)





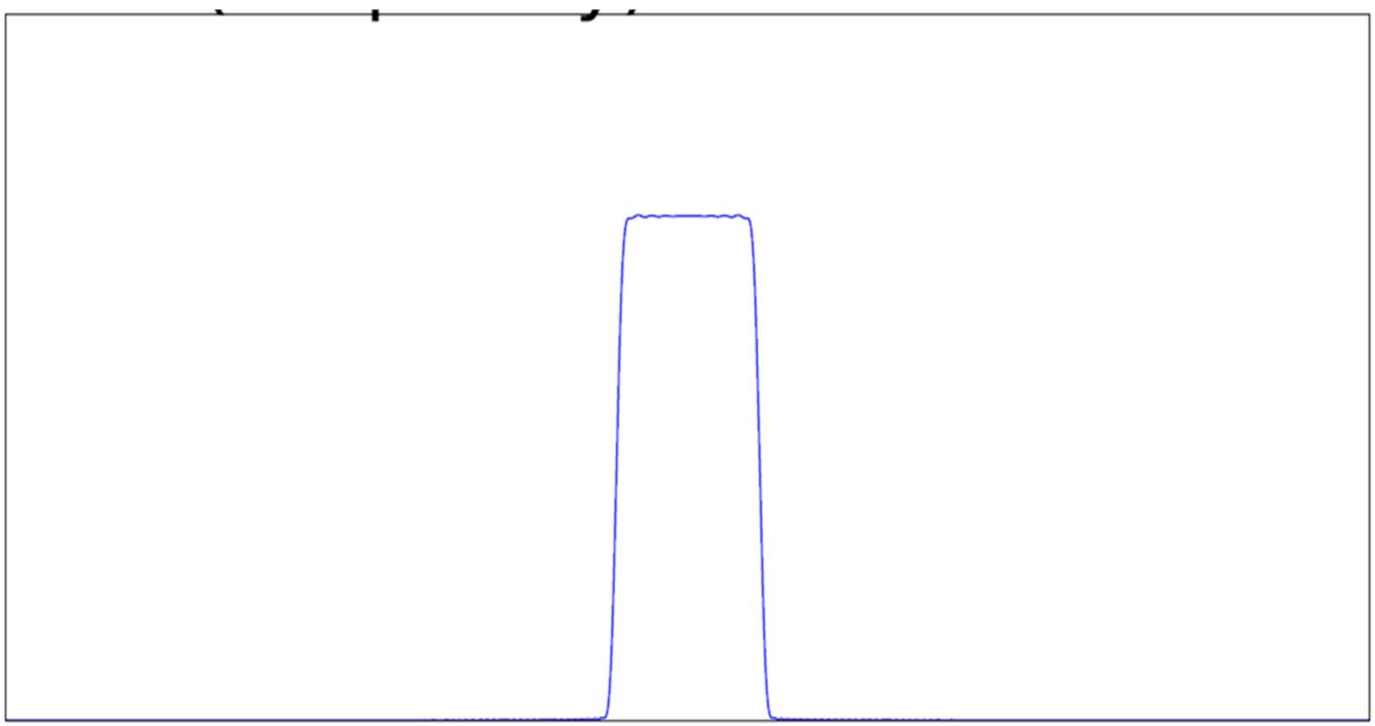
- Gaussian → Gaussian
  - Exponential decay
  - Leaking to $(\log n)^{1/2}$ buckets

Filters: Sinc × Gaussian



- Sinc × Gaussian → Boxcar*Gaussian
  - Still exponential decay
  - Almost zero leakage
  - Small support in time domain

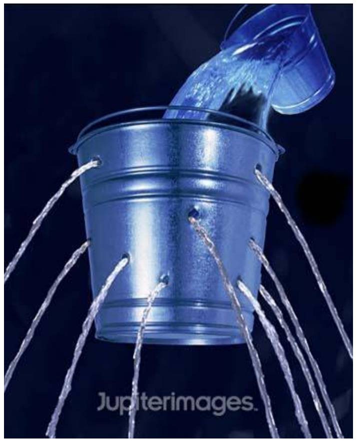
Filters: Sinc × Gaussian



- B-point FFT → Fast Bucketization
- Sinc x Gaussian → Negligible leakage

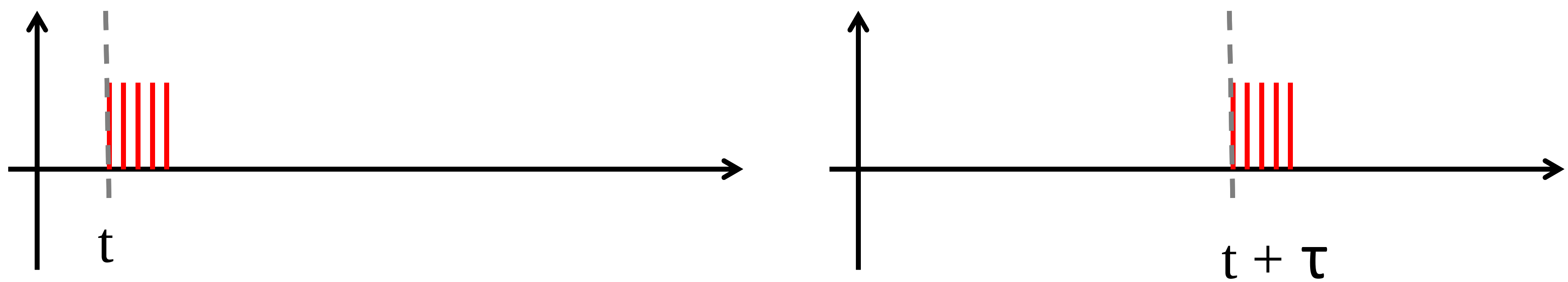
Rules of the Game

- Fast bucketization in sublinear time
- Avoid leaky buckets
- Which is the big frequency in a bucket?
- Deal with collisions



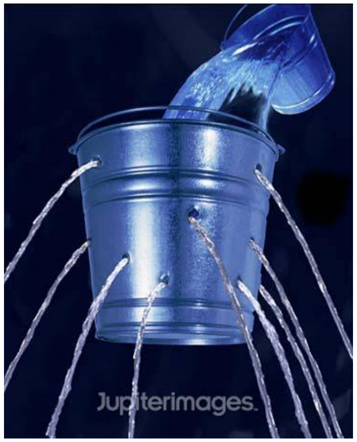
Which is the large frequency in the bucket?

- Recall: a shift in time is a phase in the frequency domain
  - FFT($\mathbf{x}^{\tau}$) = $\hat{\mathbf{x}} \times e^{-j\, 2\pi\, \tau f/n}$



- Take two B-sample FFT separated by τ
  - For each non-empty bucket, compute the phase shift
  - Phase shift of the bucket = $2\pi f_i \tau/n$ ➔ compute f_i

Rules of the Game



- Fast bucketization in sublinear time
- Avoid leaky buckets
- Which is the big frequency in a bucket?
- Deal with collisions

Dealing with Collisions

- Some Large frequencies collide:
  - Subtract and recurse
  - Small number of collisions → converges in few iterations

- Every iteration needs new random hashing:
  - Permute frequency domain: $\boldsymbol{f}' = a\boldsymbol{f} \bmod n$ (a invertible mod n)
  - Recall Scaling Property: $\mathbf{x}'(t) = \mathbf{x}(\sigma t)$ ➔ $\hat{\mathbf{x}}'(\boldsymbol{f}) = \frac{\mathbf{1}}{\sigma}\hat{\mathbf{x}}\left(\frac{\mathbf{1}}{\sigma}\boldsymbol{f}\right)$
  - For discrete case: $\mathbf{x}'(t) = \mathbf{x}(\sigma t)$ ➔ $\hat{\mathbf{x}}'(\boldsymbol{f}) = \hat{\mathbf{x}}\,(\sigma^{-1}\boldsymbol{f})$
  - Permute in time $t' = \sigma t \bmod n$ ➔ $\boldsymbol{f}' = \sigma^{-1}\boldsymbol{f} \bmod n$

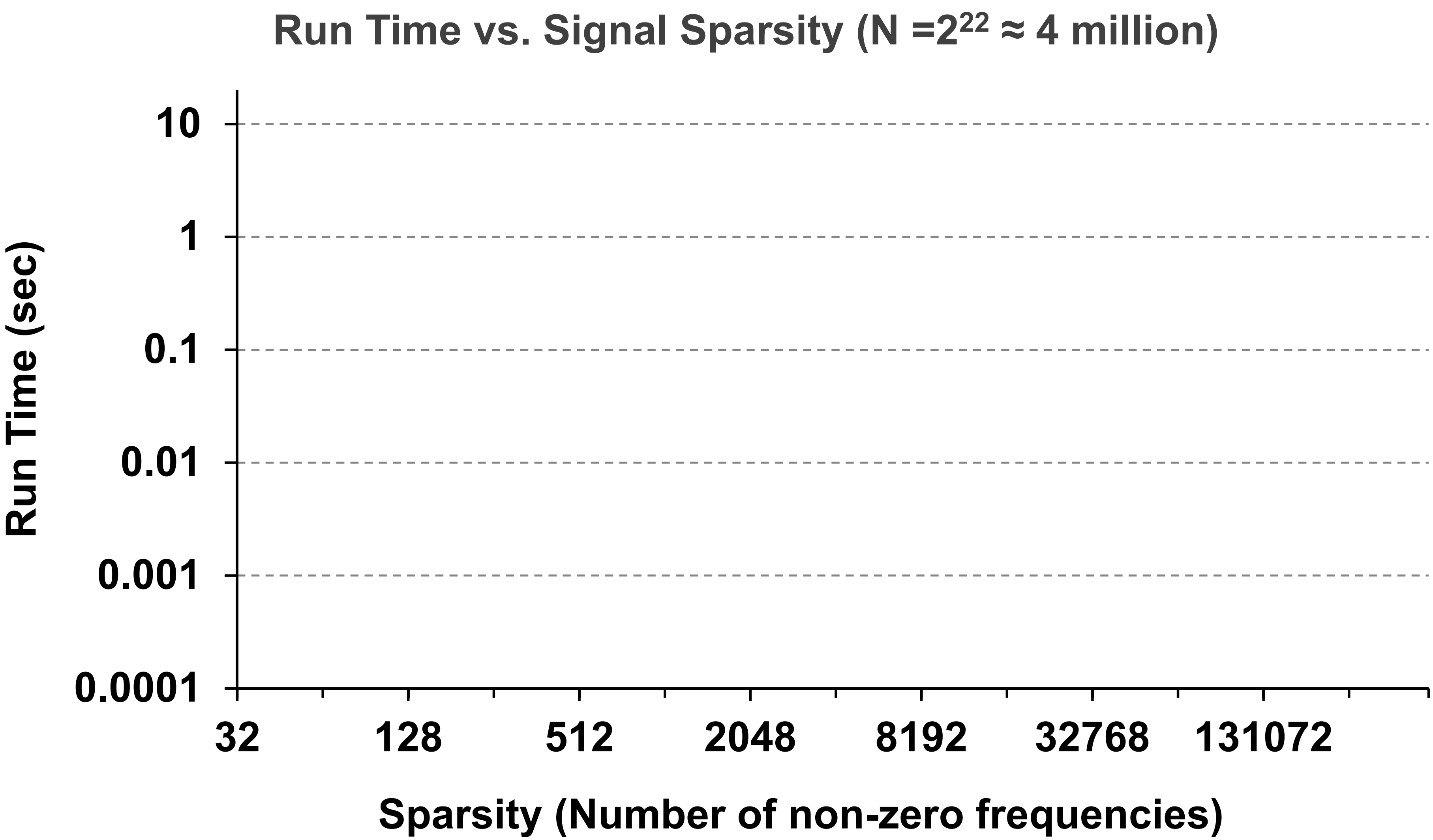
Theoretical Results

- For a signal of size n with k large frequencies

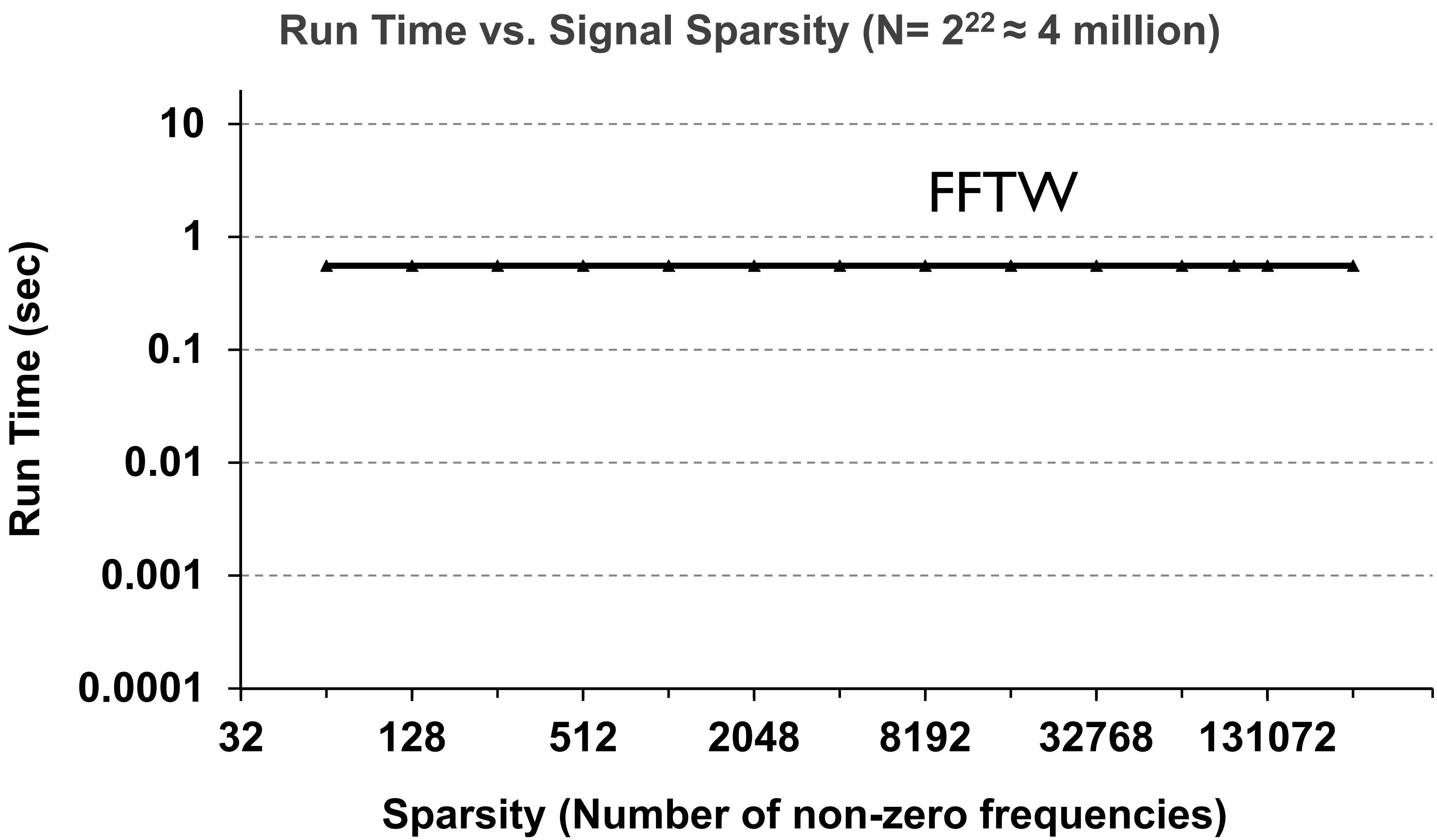
- Prior work on sparse FFT
  - $O(k \log^c n)$ for some c is about 4 [GMS05, Iwen'10]
  - Improves over FFT for $k << n/\log^3 n$

- Our results [SODA'12], [STOC'12]
  - Exactly k-sparse case : $O(k \log n)$
    - Optimal if FFT is optimal
  - Approximately k-sparse case $O(k \log(n) \log(n/k))$
    - Improves over FFT for any $k = o(n)$

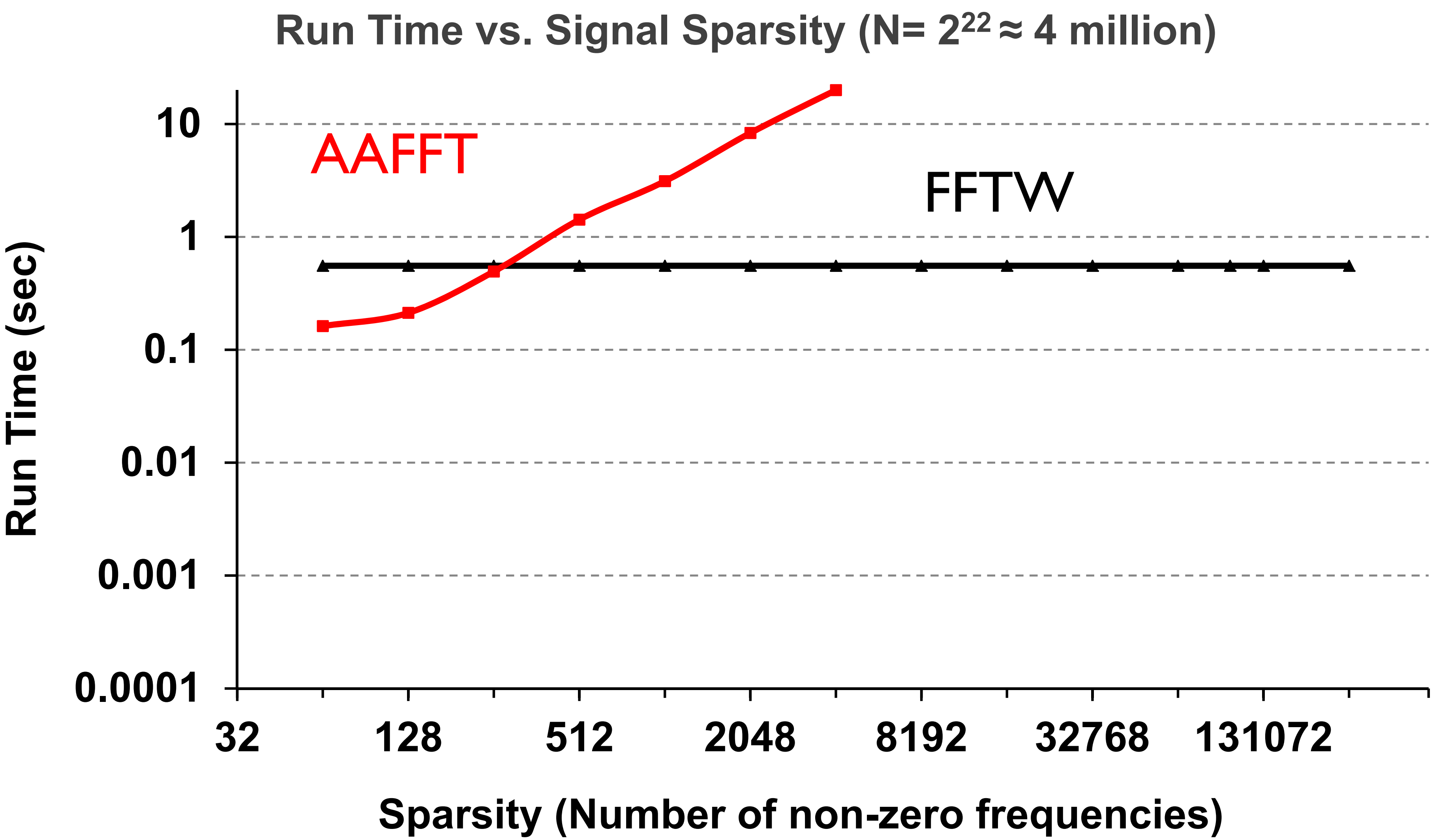
Simulation Results



Simulation Results



Simulation Results



Simulation Results