Lossy compression algorithms for floating-point data

JSM 2017

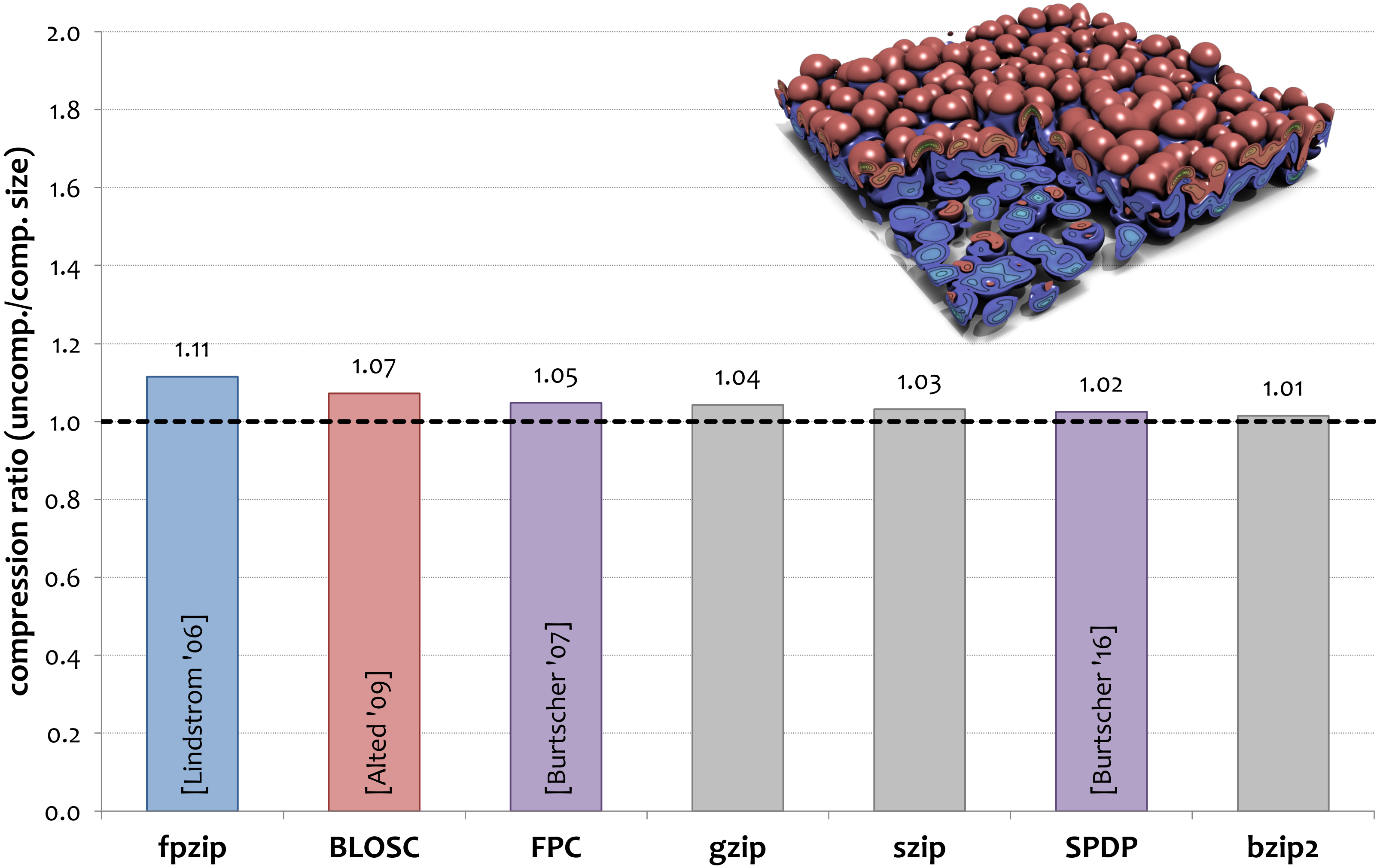
Peter Lindstrom
pl@llnl.gov

July 31, 2017

LLNL-PRES-736379
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Numerical data is challenging to compress losslessly



Lossy compression enables greater reduction, but is often met with skepticism by scientists

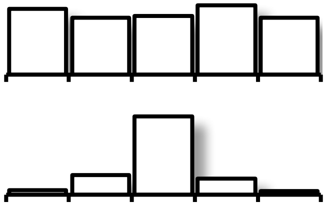
- Large improvements in compression possible by allowing even small errors
  - Least significant floating-point bits are effectively random noise
  - Most compressors support relative or absolute error tolerances

- Compressors must be cognizant of how compression errors propagate in data analysis
  - Biased error (ideally zero mean)
  - Correlation of error with function (ideally independent)
  - Autocorrelation of error (ideally uncorrelated)
  - Spectral properties of error (ideally white noise)
  - Distribution of error (e.g. uniform, normal, Laplace, …)
  - Impact on statistical quantities like extrema, mean/median, moments, …
  - Impact on differential quantities like spatial & temporal derivatives

- This talk will examine error distributions for several compressors

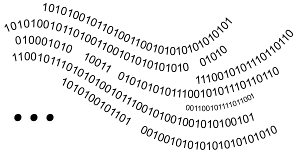
Numerical data compression usually involves three steps

- 1. **Decorrelate** data to make it more compressible
  - E.g. prediction, fitting, transformation, decomposition, …
  - Make the data sparse in some alternative representation
  - Small values, repeated patterns are easier to compress



- 2. **Approximate** (for lossy compression)
  - E.g. scalar/vector quantization, truncation, thresholding, …
  - Discard unimportant information to avoid encoding it

- 3. **Encode** remaining information losslessly
  - E.g. Huffman, arithmetic, universal, run-length, dictionary, …



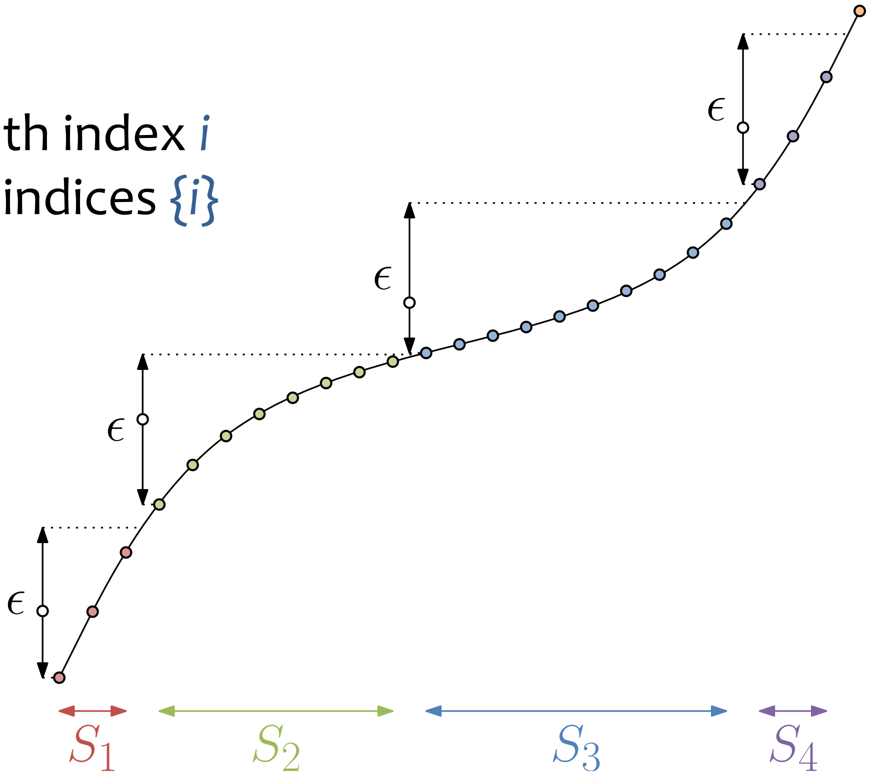
Case study: 8 lossy floating-point compressors in 8 minutes!

1. SQ: adaptive scalar quantization [Iverson et al. 2013]
2. HVQ: hierarchical vector quantization [Schneider & Westermann 2003]
3. SZ: error-bounded polynomial prediction [Di & Cappello 2016]
4. fpzip: lossless/lossy predictive coding [Lindstrom & Isenburg 2006]
5. ZFP: block transform with embedded coding [Lindstrom 2014]
6. VAPOR: wavelet transform & thresholding [Clyne et al. 2007]
7. Tucker: tensor decomposition & thresholding [Ballester & Pajarola 2016]
8. ISABELA: sorting and spline fitting [Lakshminarasimhan et al. 2013]

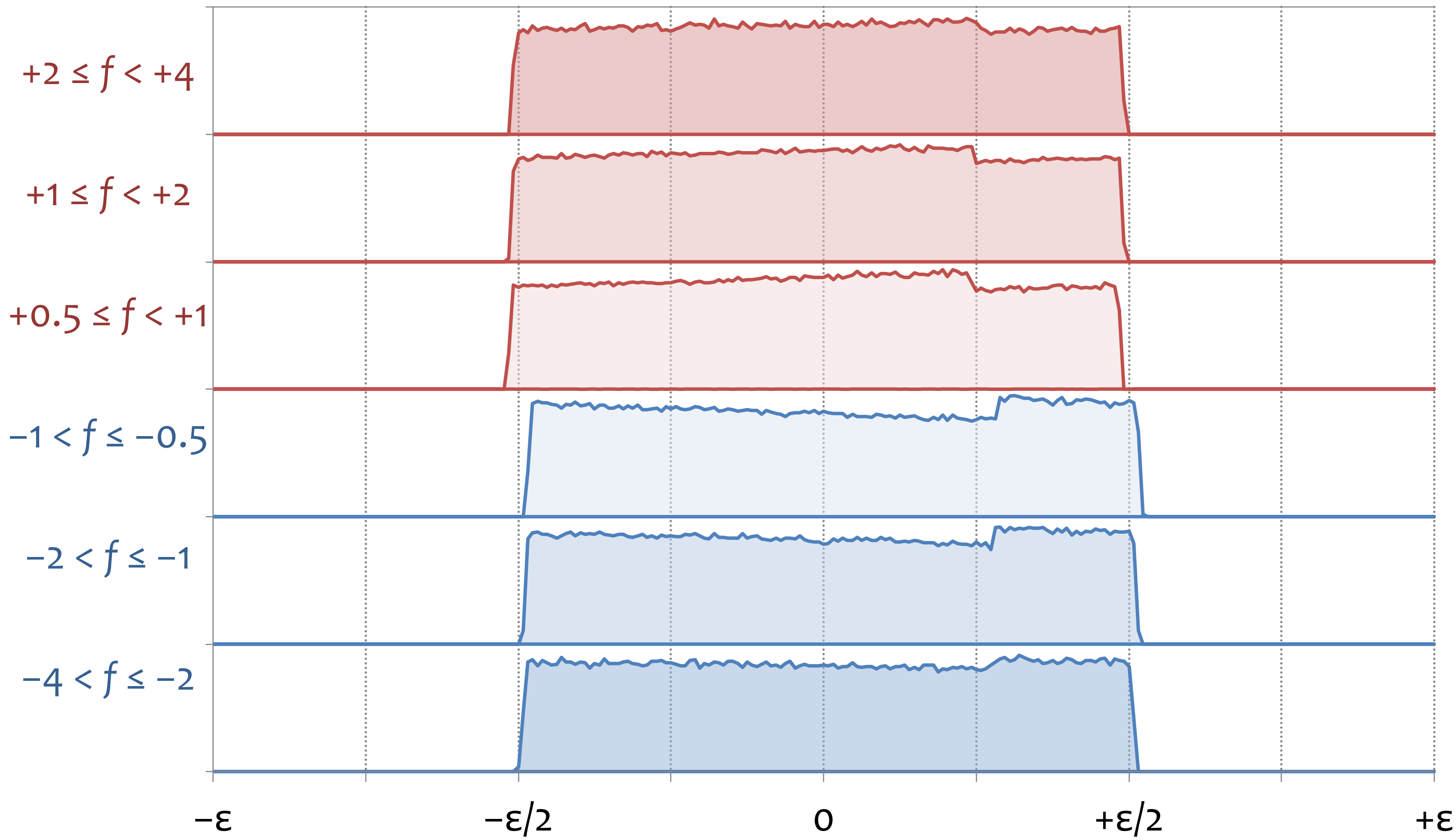
Challenge: Compress 3D scalar field, f(x, y, z), defined on uniform Cartesian grid

SQ: Adaptive, error-bounded scalar quantization

- [SQ] algorithm partitions data into ε-sized ranges
  - Sort data on function value
  - Greedily grow set S_i as long as max S_i – min S_i ≤ ε
  - Use as prototype p_i = mean S_i
    - Minimizes RMS error
  - Replace values assigned to set S_i with index i
  - LZMA compress codebook $\{p_i\}$ and indices $\{i\}$

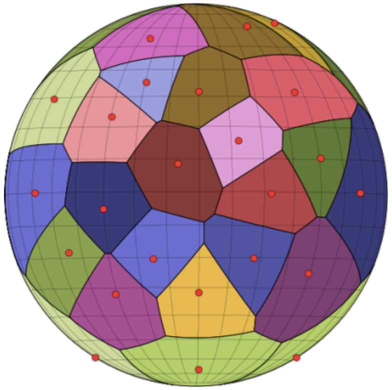


SQ error distribution is nearly uniform but overly conservative



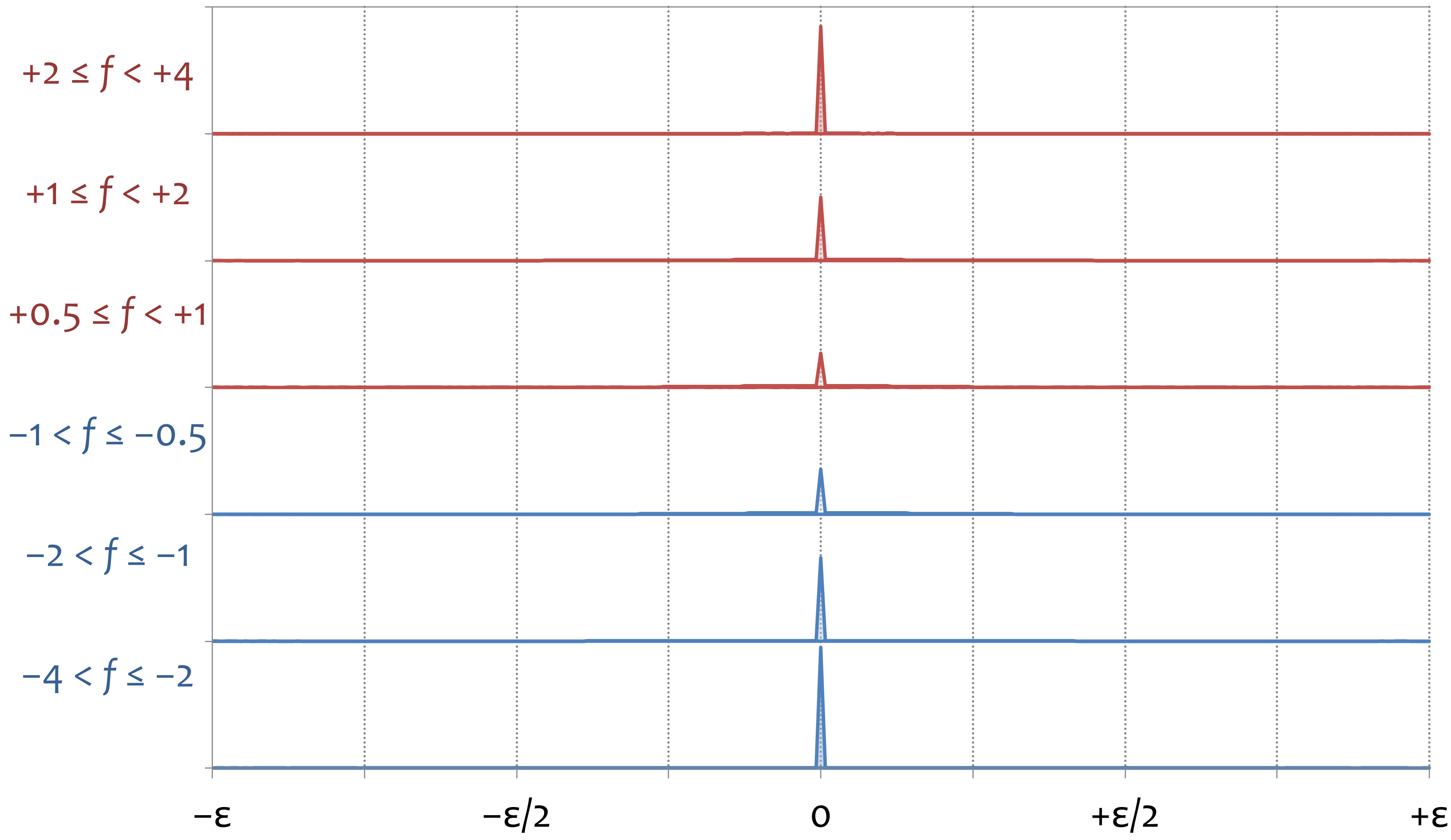
HVQ: Hierarchical Vector Quantization

- Similar to scalar quantization, but applied to multi-component vectors
  - E.g. vector/tensor fields, multiple correlated fields, blocks of values, …
  - Can be done non-uniformly in both domain and range
- Hierarchical VQ [HVQ] uses different codebook on each level
  - Vectors formed by 4 × 4 × 4 blocks of values
    - Next level given by block averages
  - Codebook is generated using Lloyd relaxation
    - Randomly select initial prototypes
    - Partition data by closest prototype
    - Replace prototype with mean/medoid/Voronoi centroid



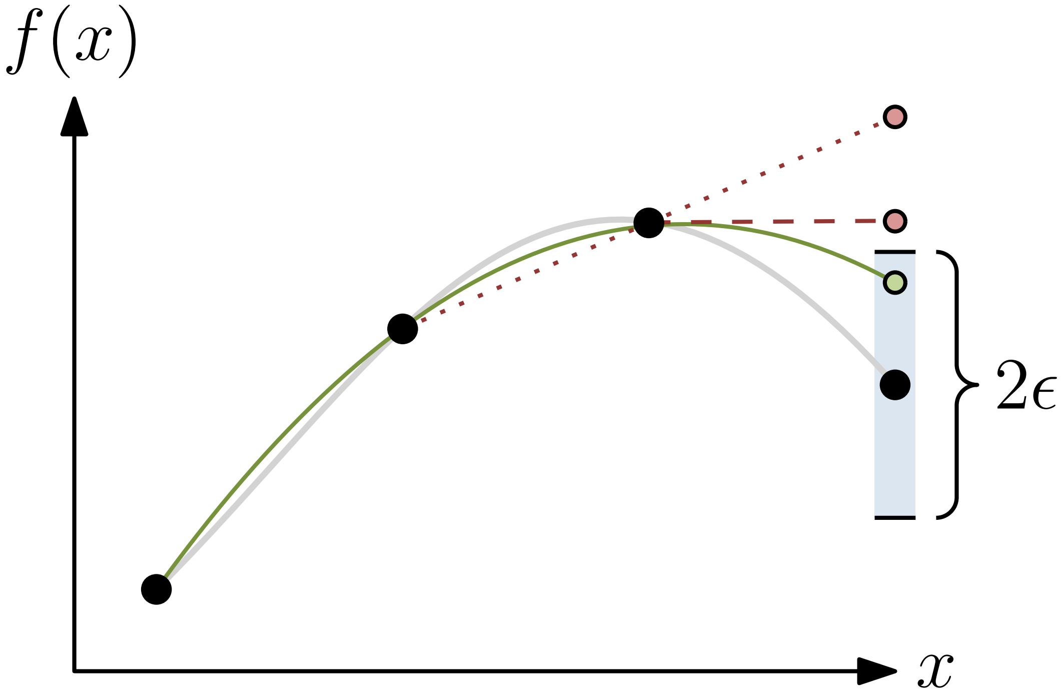
- Most effective for low-precision data like 8-bit RGB
  - Codebook size, compute time become prohibitive for higher precision

VQ errors are difficult to bound due to difficulty of creating good codebook

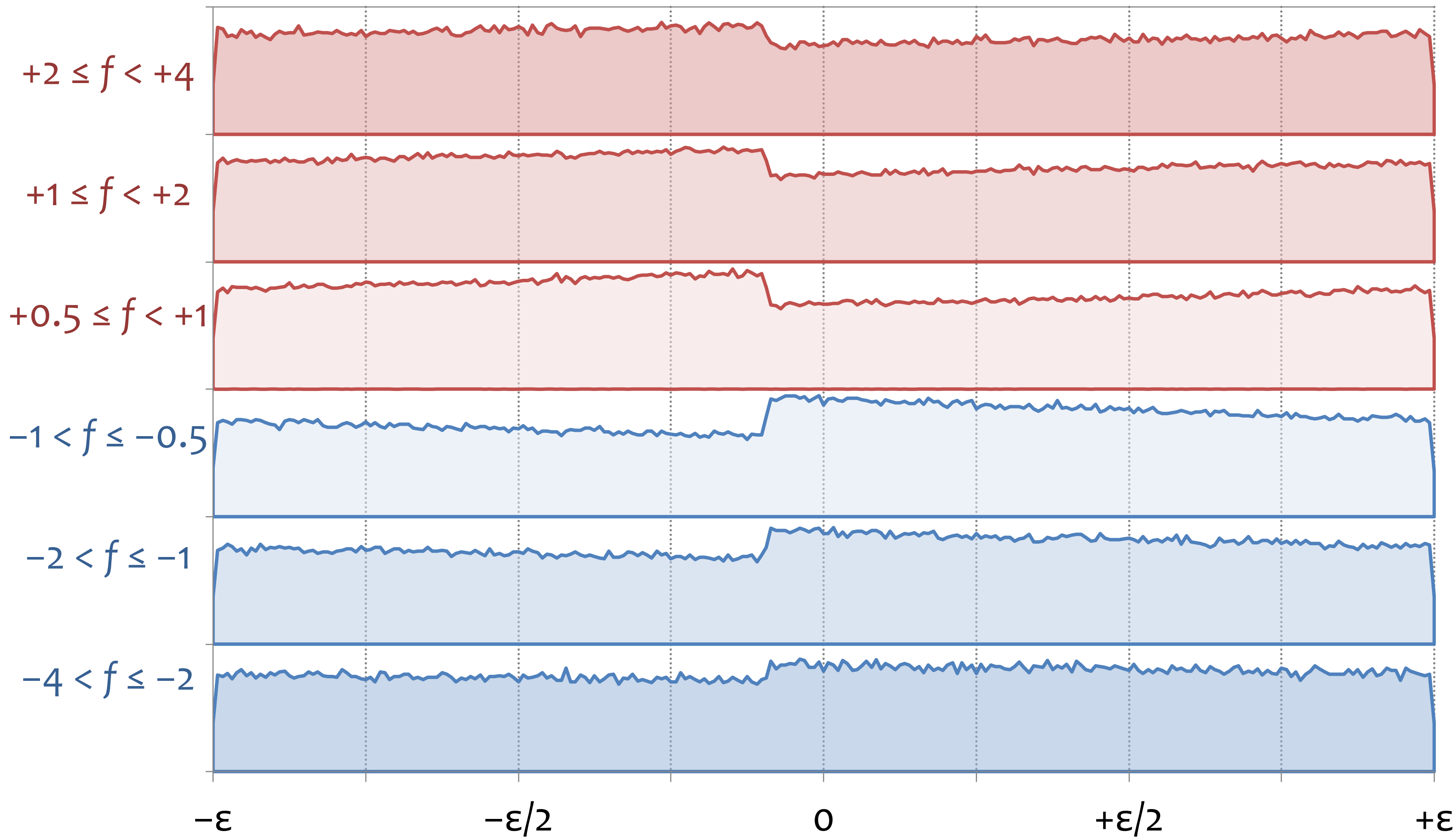


SZ: Polynomial prediction extrapolates from past data points

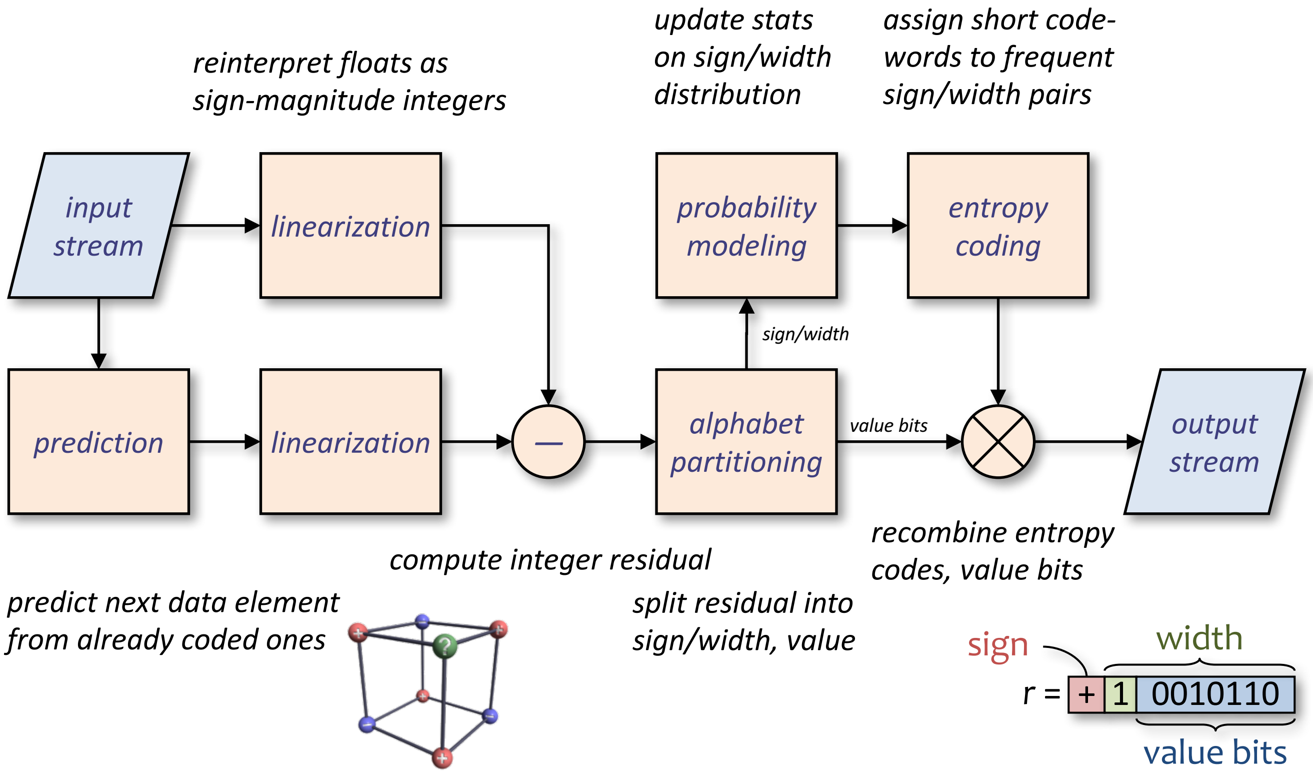
- Polynomial of degree *n* – 1 predicts next value from last *n* transmitted values
  - Use best of three predictors: constant, linear, quadratic
  - “Mispredictions” outside of tolerance ±ε are corrected



SZ error distribution is approximately uniform and spans full tolerance



fpzip: Lossless mode combines multi-dimensional prediction with entropy coding



reinterpret floats as sign-magnitude integers

update stats on sign/width distribution

assign short code-words to frequent sign/width pairs

input stream

linearization

probability modeling

entropy coding

sign/width

prediction

linearization

—

alphabet partitioning

value bits

output stream

compute integer residual

recombine entropy codes, value bits

predict next data element from already coded ones

split residual into sign/width, value

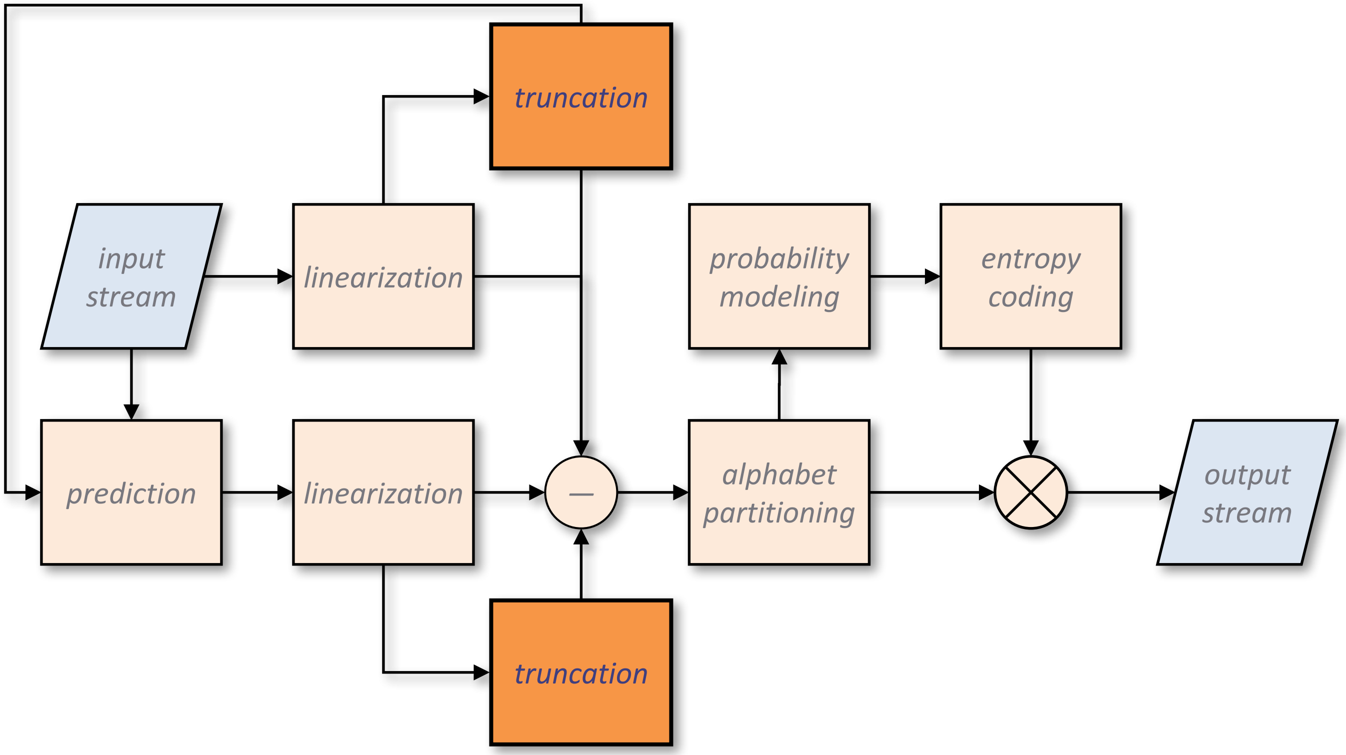
sign

width

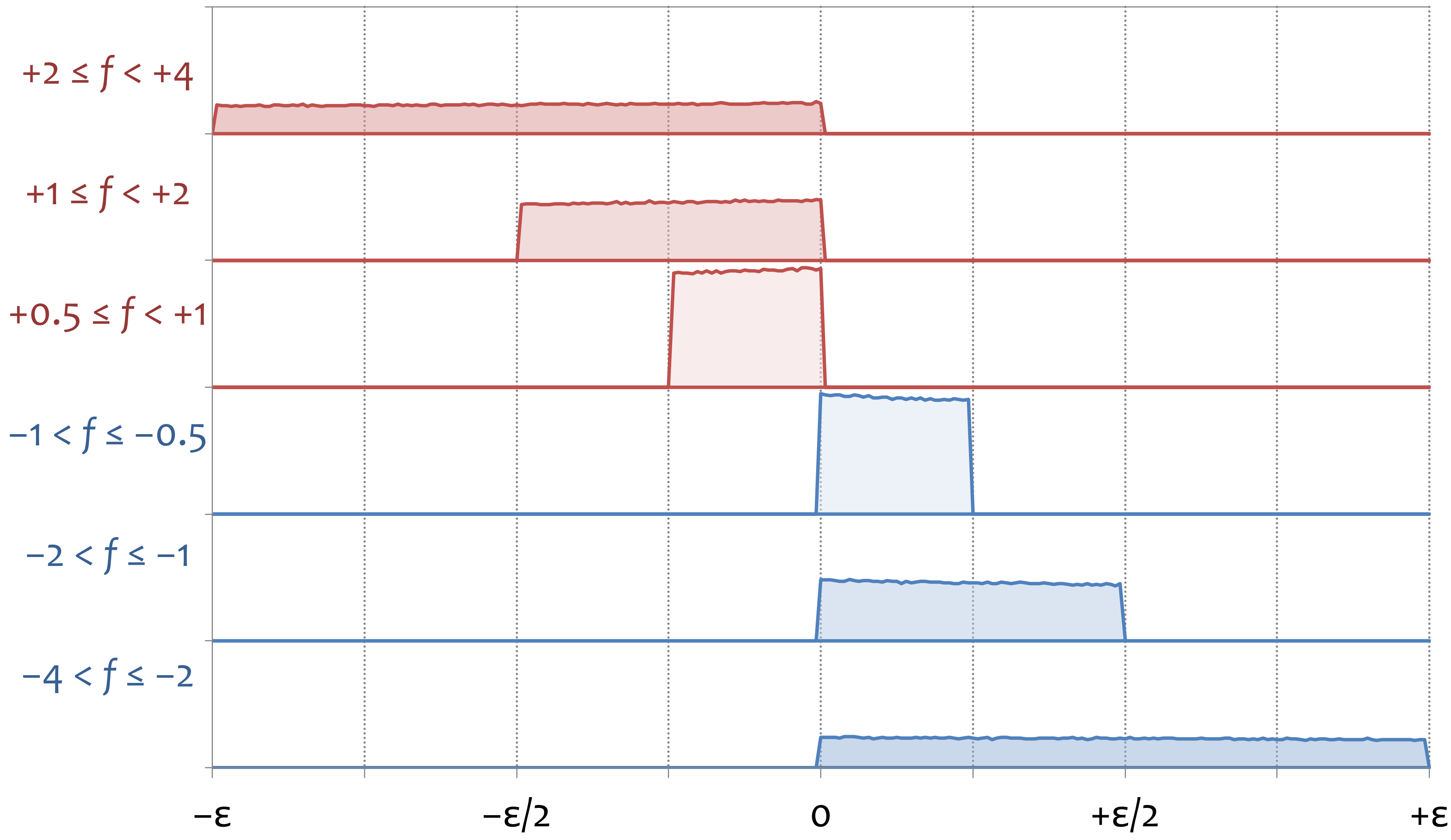
r = + 1 0010110

value bits

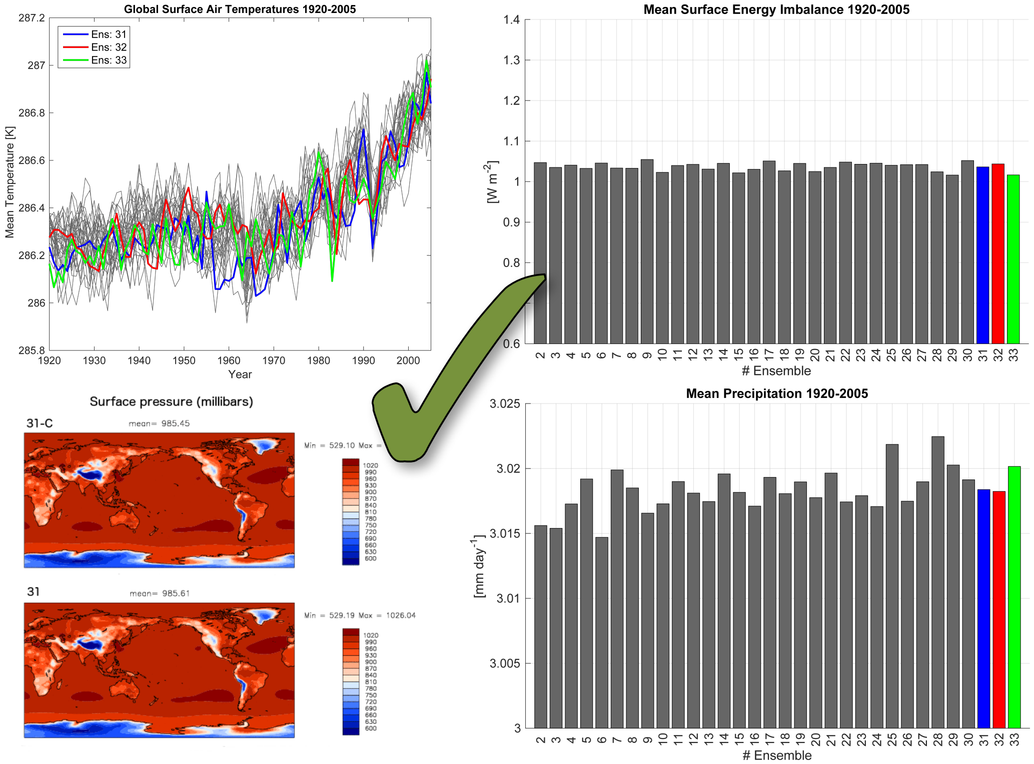
fpzip: Lossy mode truncates (zeros) least significant bits, then compresses losslessly



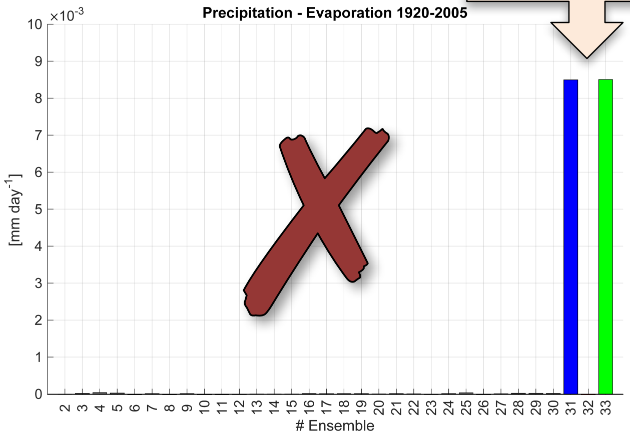
fpzip error distribution is dependent on function value *f* and is highly biased



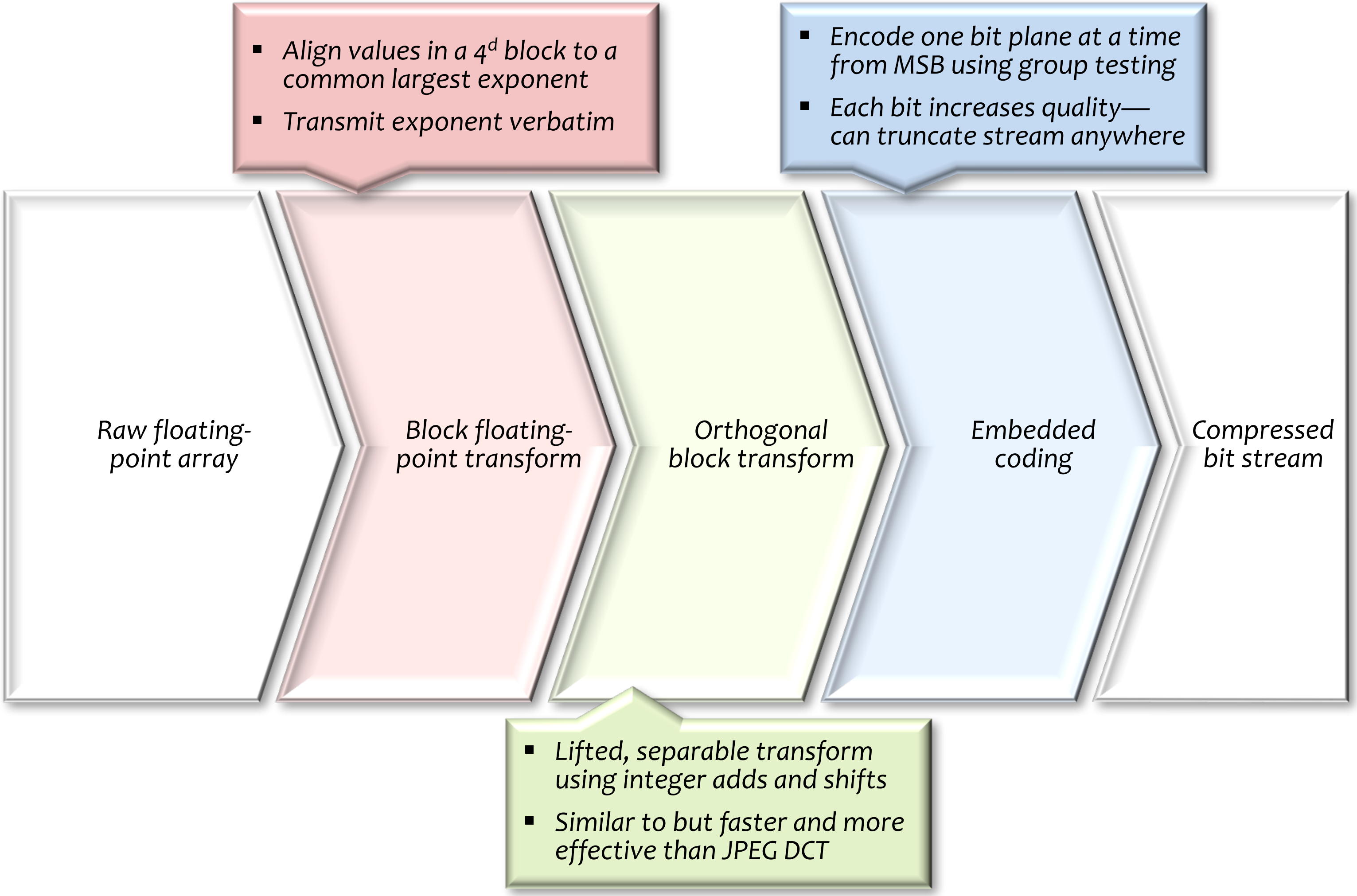
fpzip systematic rounding toward zero leads to occasional issues in climate data analysis



#31, #33 compressed ensemble members



ZFP: Compressed floating-point arrays that support random access and error tolerances



Raw floating-point array

Block floating-point transform

- *Align values in a 4^d block to a common largest exponent*
- *Transmit exponent verbatim*

Orthogonal block transform

- *Lifted, separable transform using integer adds and shifts*
- *Similar to but faster and more effective than JPEG DCT*

Embedded coding

- *Encode one bit plane at a time from MSB using group testing*
- *Each bit increases quality—can truncate stream anywhere*

Compressed bit stream

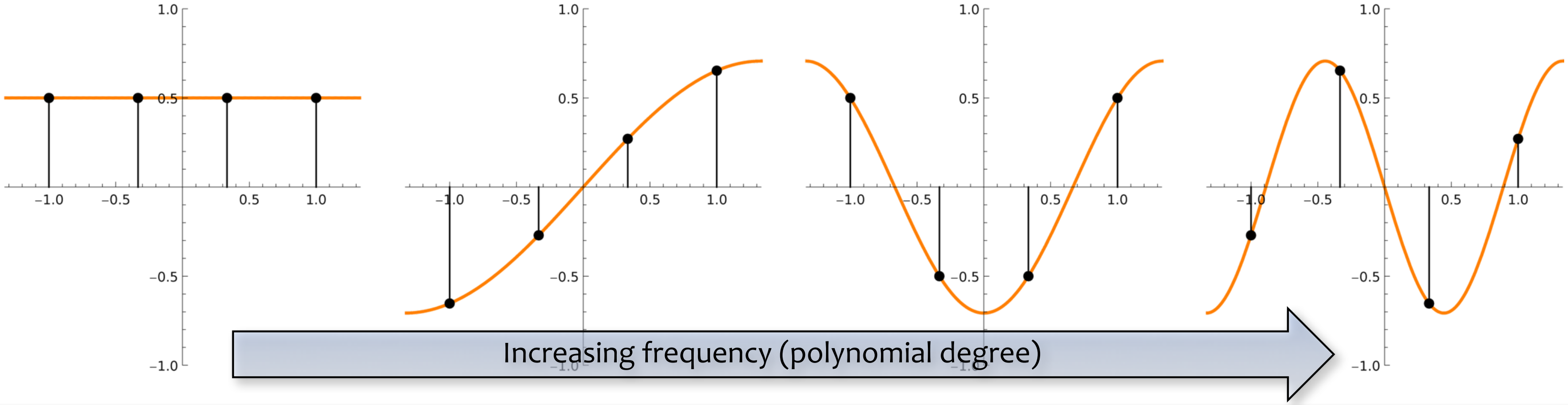
ZFP decorrelates *d*-dimensional block of 4^d values using an orthogonal transform

$$\underbrace{\begin{pmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \hat{f}_4 \end{pmatrix}}_{\textit{coefficients}} = \underbrace{\frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ c & s & -s & -c \\ 1 & -1 & -1 & 1 \\ s & -c & c & -s \end{pmatrix}}_{\textit{orthogonal transform}} \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}}_{\textit{block}}$$

Free parameter t

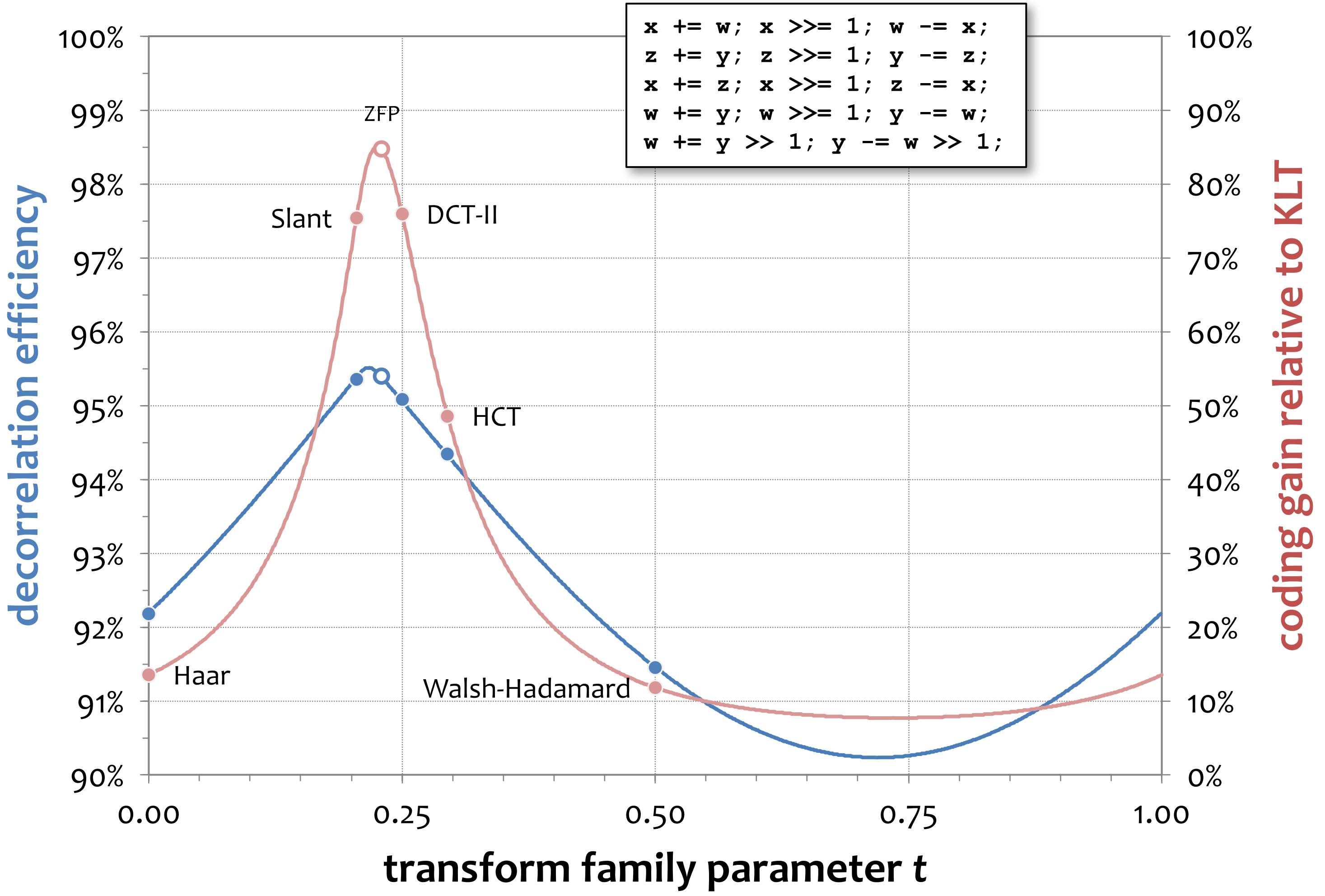
$$s = \sqrt{2}\sin\frac{\pi}{2}t \qquad c = \sqrt{2}\cos\frac{\pi}{2}t$$

Basis functions for t = 1/4



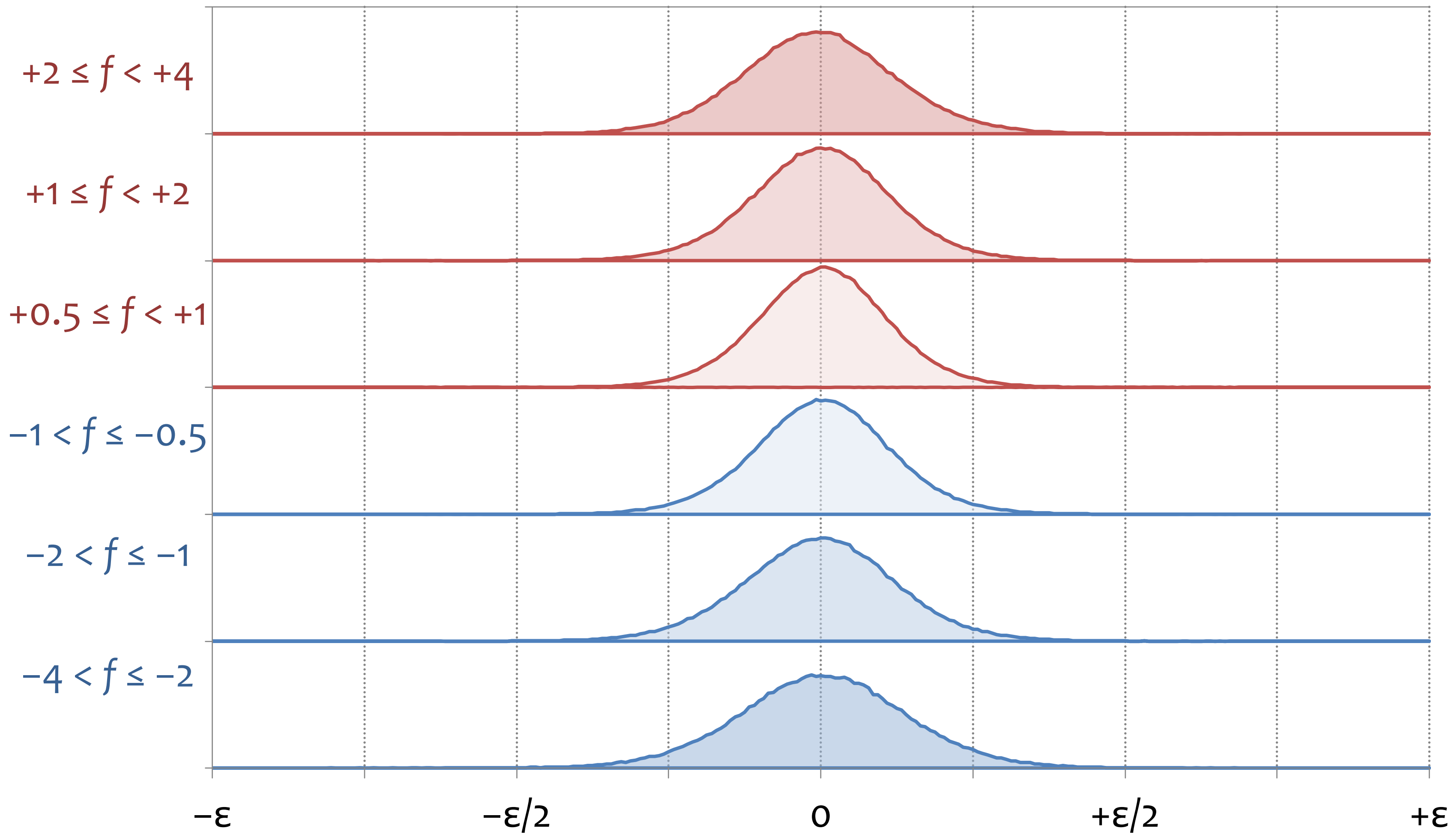
Increasing frequency (polynomial degree)

ZFP's integer transform is efficient, effective, and well-suited for h/w implementation



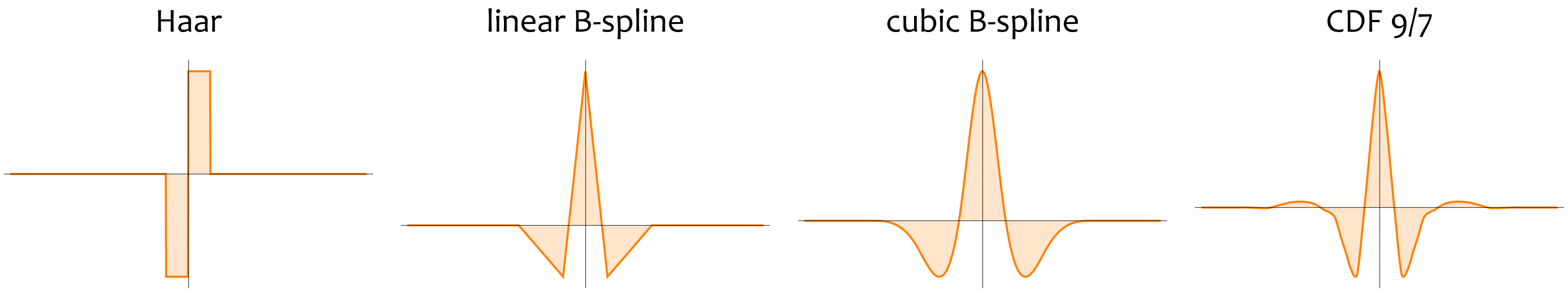
```
x += w; x >>= 1; w -= x;
z += y; z >>= 1; y -= z;
x += z; x >>= 1; z -= x;
w += y; w >>= 1; y -= w;
w += y >> 1; y -= w >> 1;
```

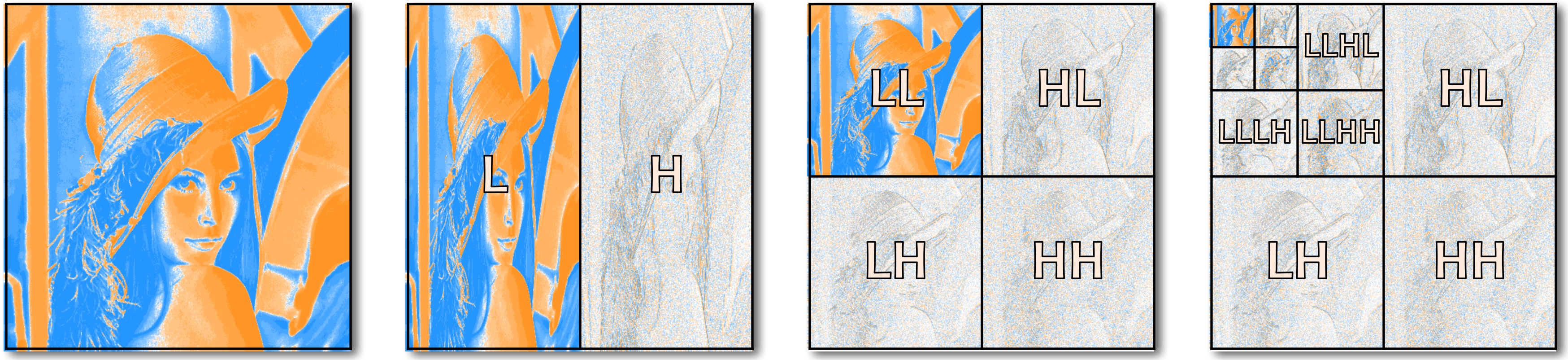
ZFP error distribution is normal due to linear transform of iid. errors (central limit theorem)



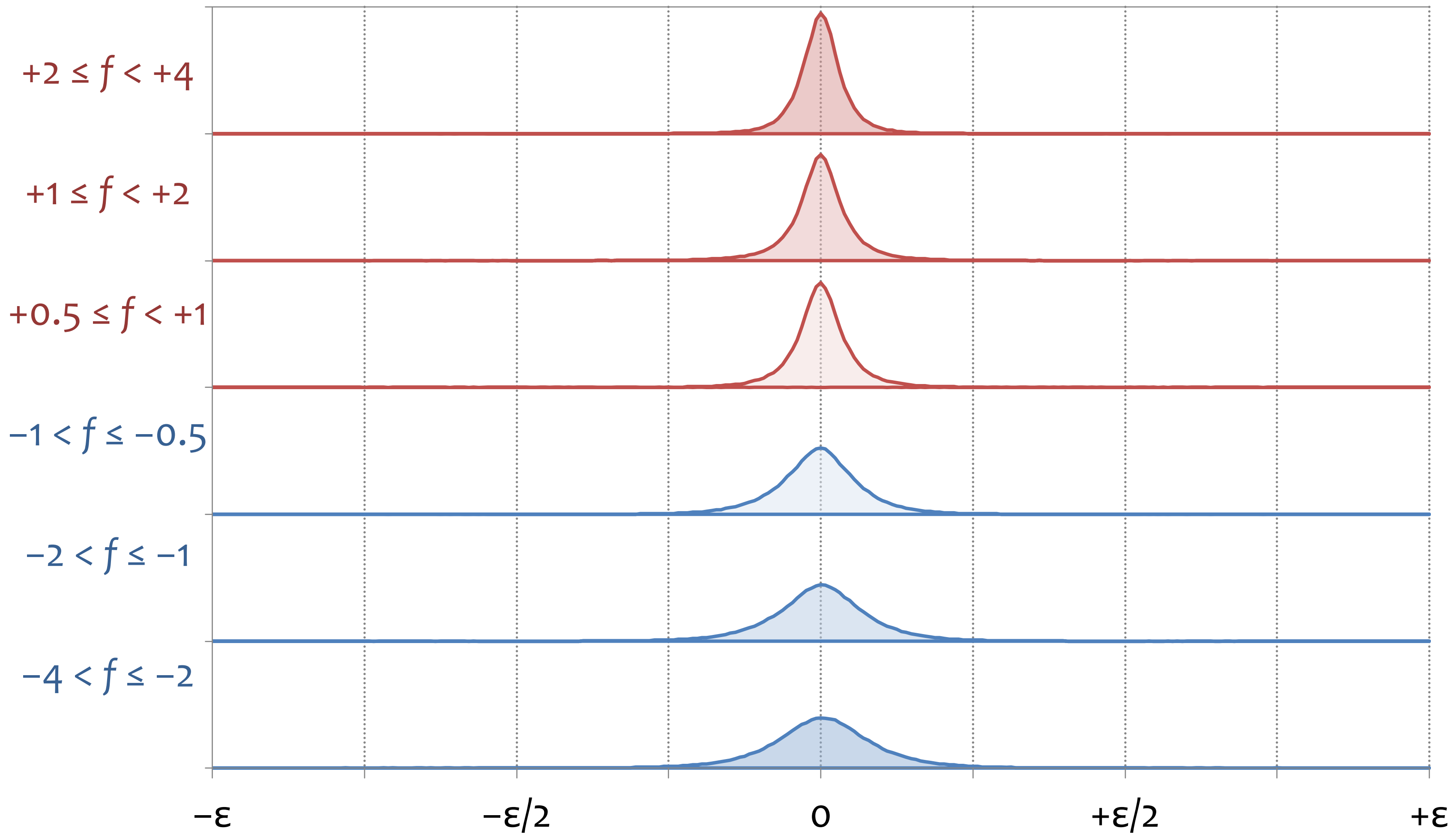
VAPOR: Discrete wavelet transform with coefficient thresholding

- Basis functions are given by translations and dilations of single mother wavelet





VAPOR wavelet errors are difficult to bound due to cascading effects



Tucker: Generalization of SVD using Tucker tensor decomposition, core tensor truncation

- 2D structured grid data can be approximated via truncated SVD

$$\mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n) = \Sigma = U^T A V = \mathrm{unvec}\big((V \otimes U)^T \mathrm{vec}(A)\big)$$

  – Singular value matrix, Σ, is **diagonal** but singular vectors, *U* and *V*, are **data-dependent**
    - *U* and *V* are expensive to encode for 2D data

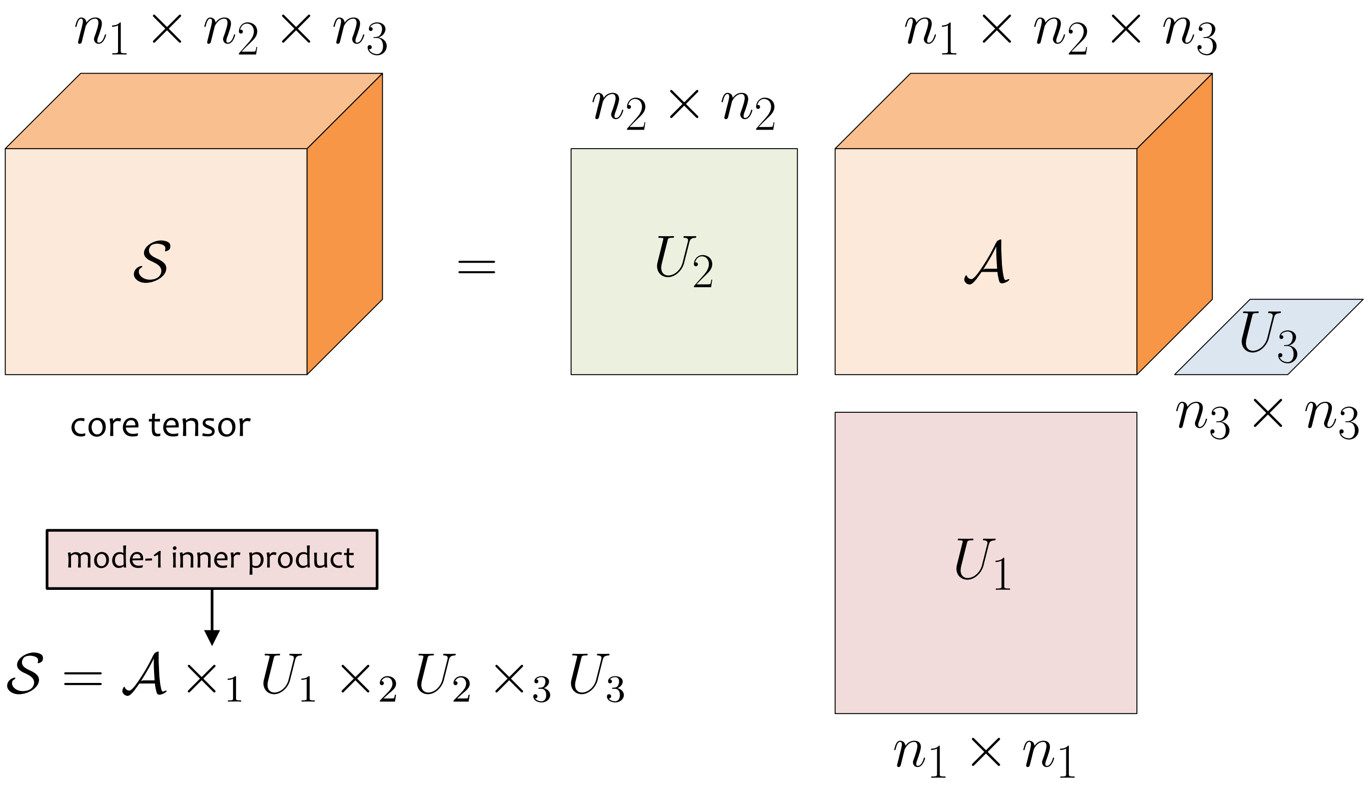
  – *A* can be optimally approximated in the L_2 sense by discarding smallest singular values

- *d*-dimensional structured grid data can be approximated via tensor decomposition

$$\mathcal{S} = \mathcal{A} \times_1 U \times_2 V \times_3 W = \mathrm{unvec}\big((W \otimes V \otimes U)^T \mathrm{vec}(\mathcal{A})\big)$$

  – Unlike in SVD, core tensor, *S*, is not diagonal, but large values appear in “hot corner”
    - *U*, *V*, and *W* matrices are relatively cheap to encode for 3D data

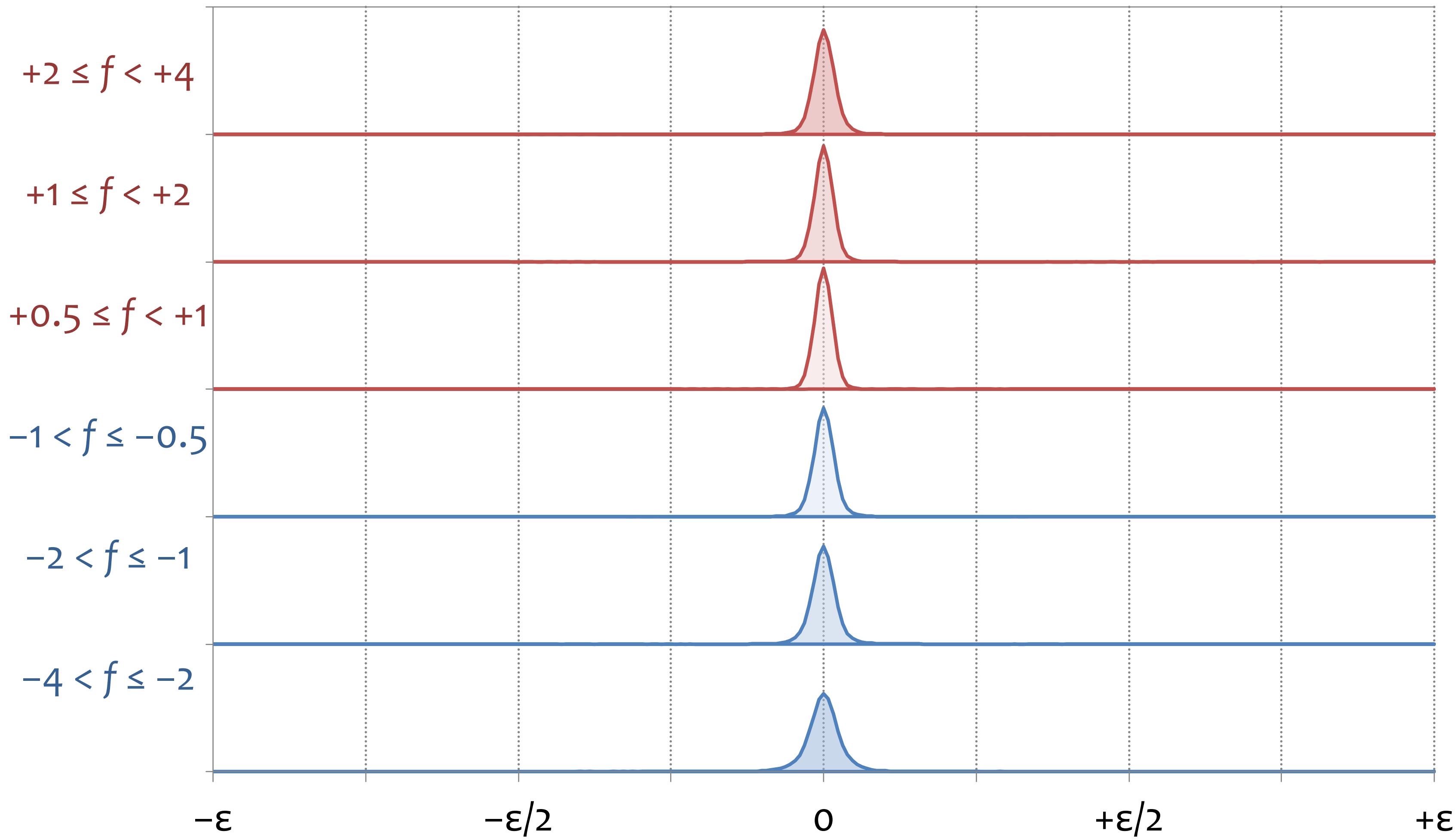
As in SVD, truncated core tensor & factor matrices yield “best” low-rank approximation



mode-1 inner product

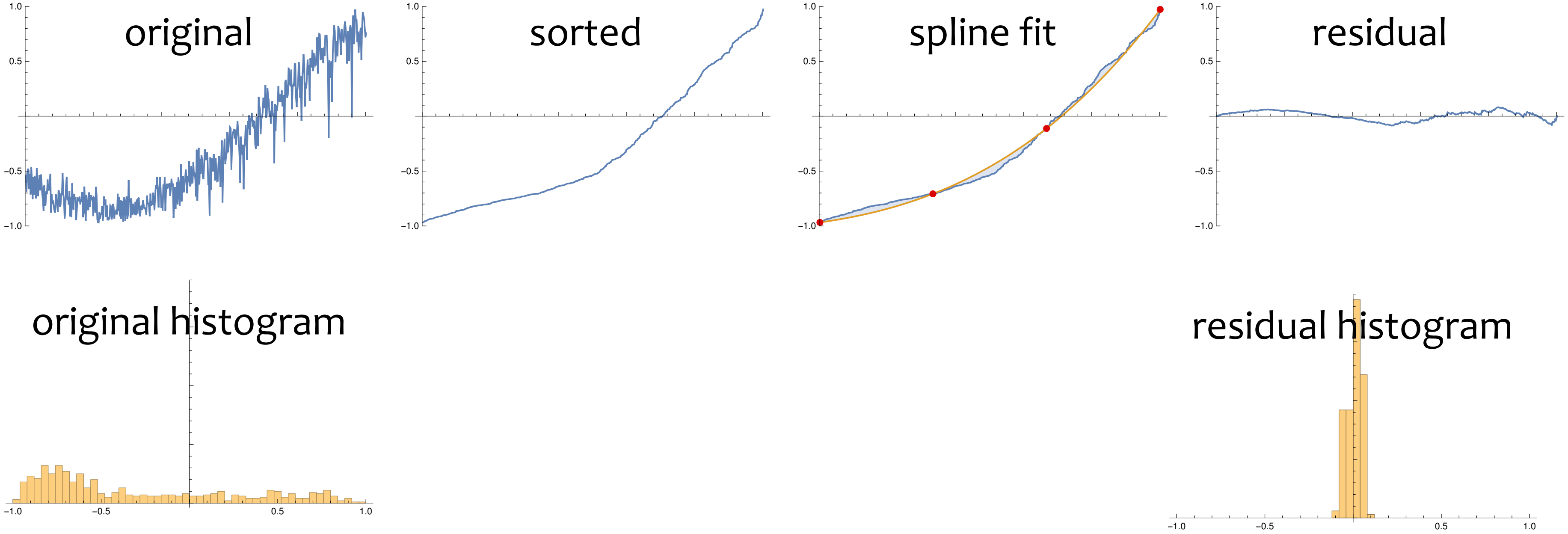
$$\mathcal{S} = \mathcal{A} \times_1 U_1 \times_2 U_2 \times_3 U_3$$

Like wavelets, Tucker tensor decomposition errors are difficult to bound tightly

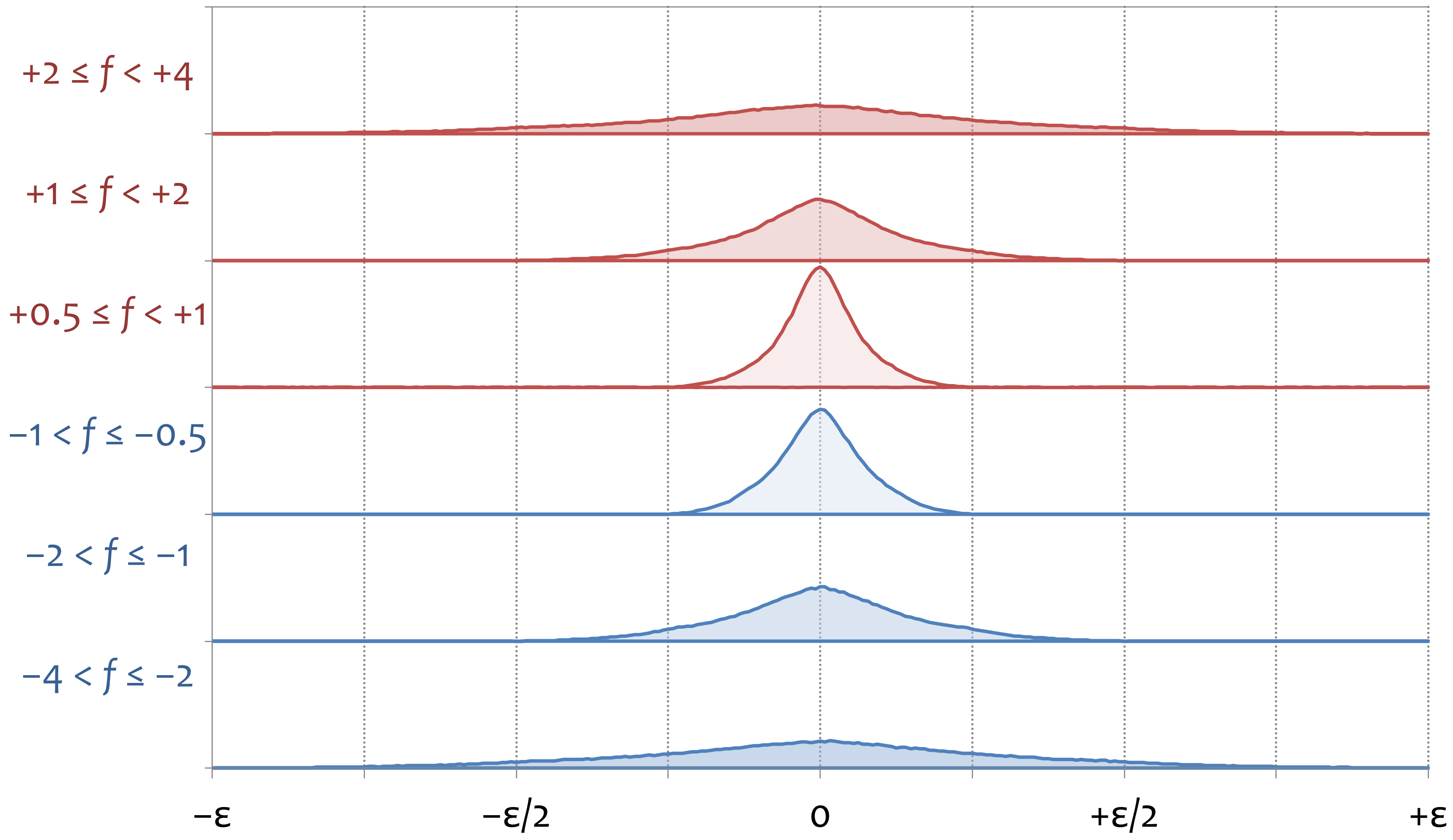


ISABELA: Sorting and spline fitting enables compression of even the noisiest data sets

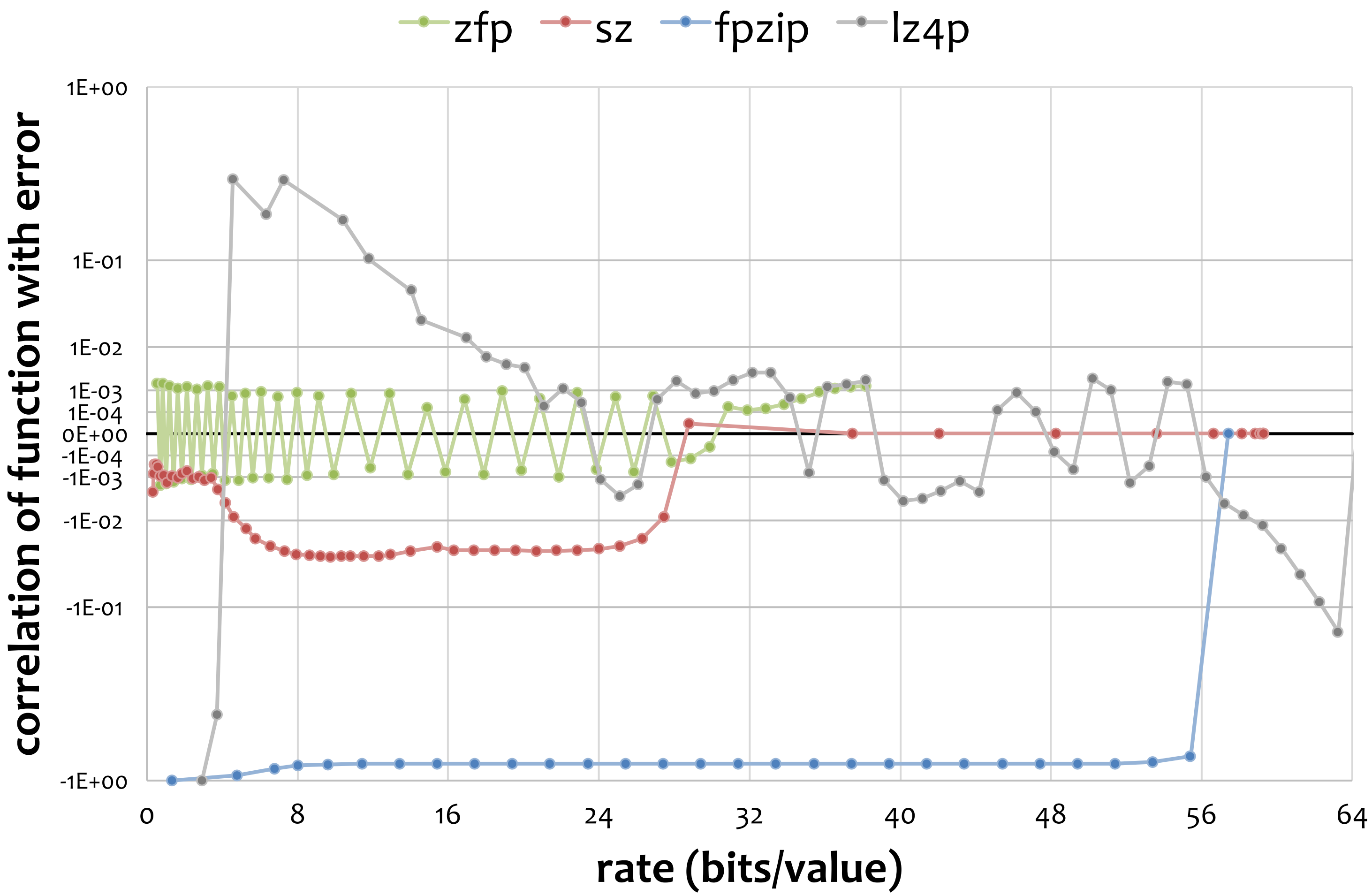
- Most compression techniques fail miserably on noisy/unstructured data
- [ISABELA]: Sort noisy data, encode permutation, fit smooth sorted signal



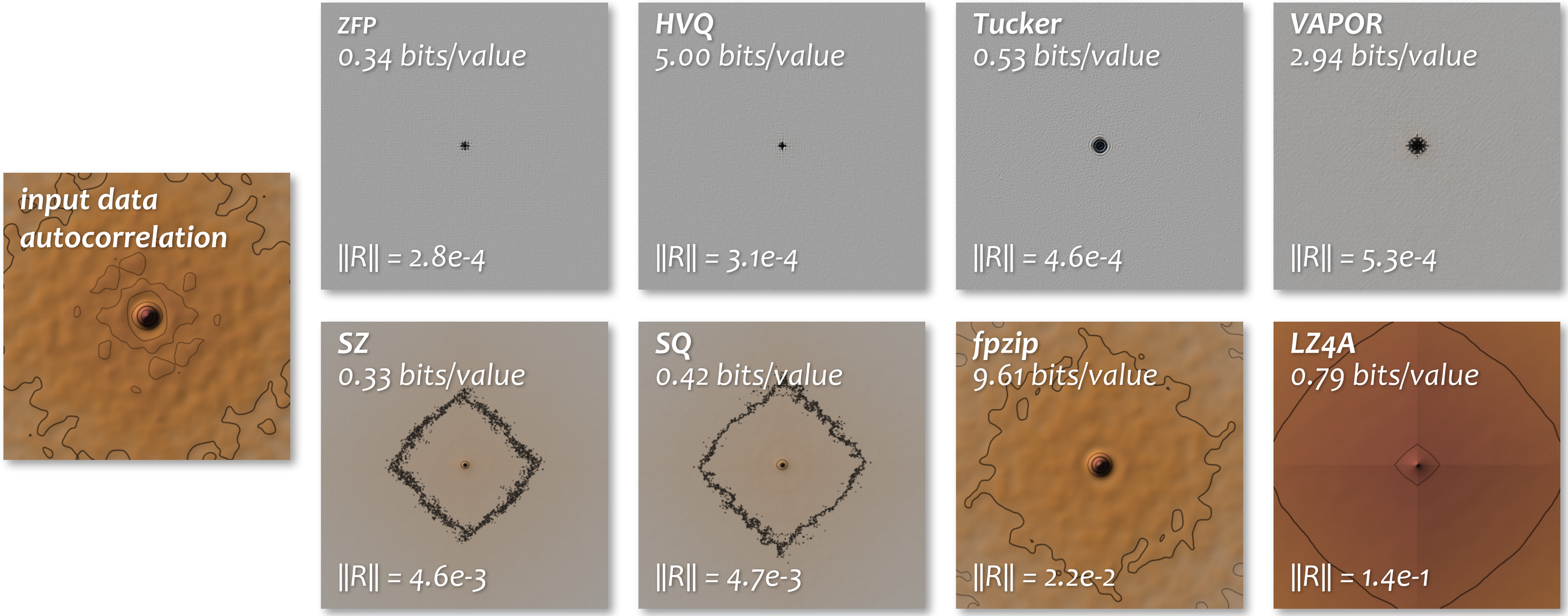
Like fpzip, ISABELA bounds relative errors, but without bias



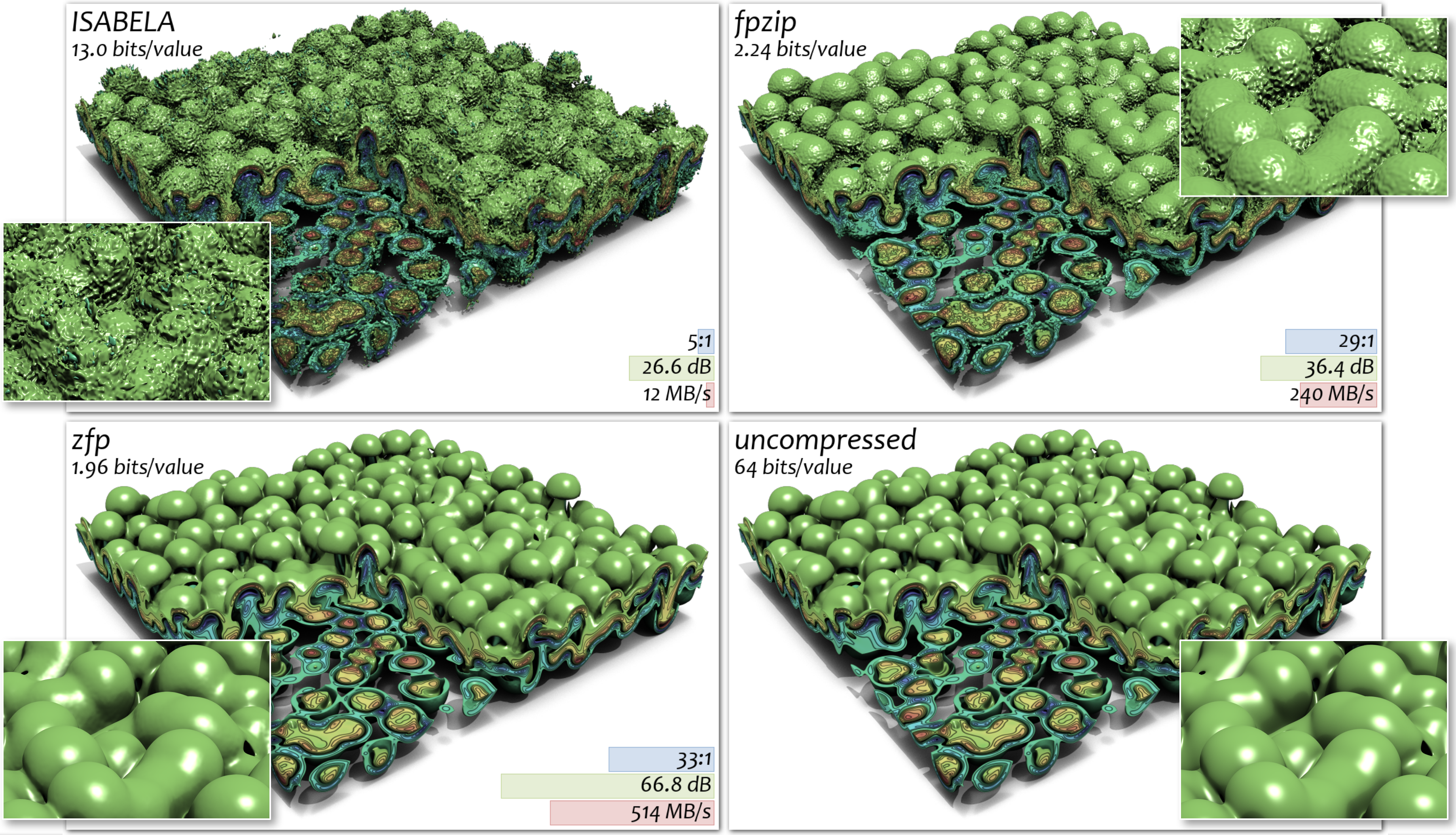
ZFP and SZ decorrelate error with function



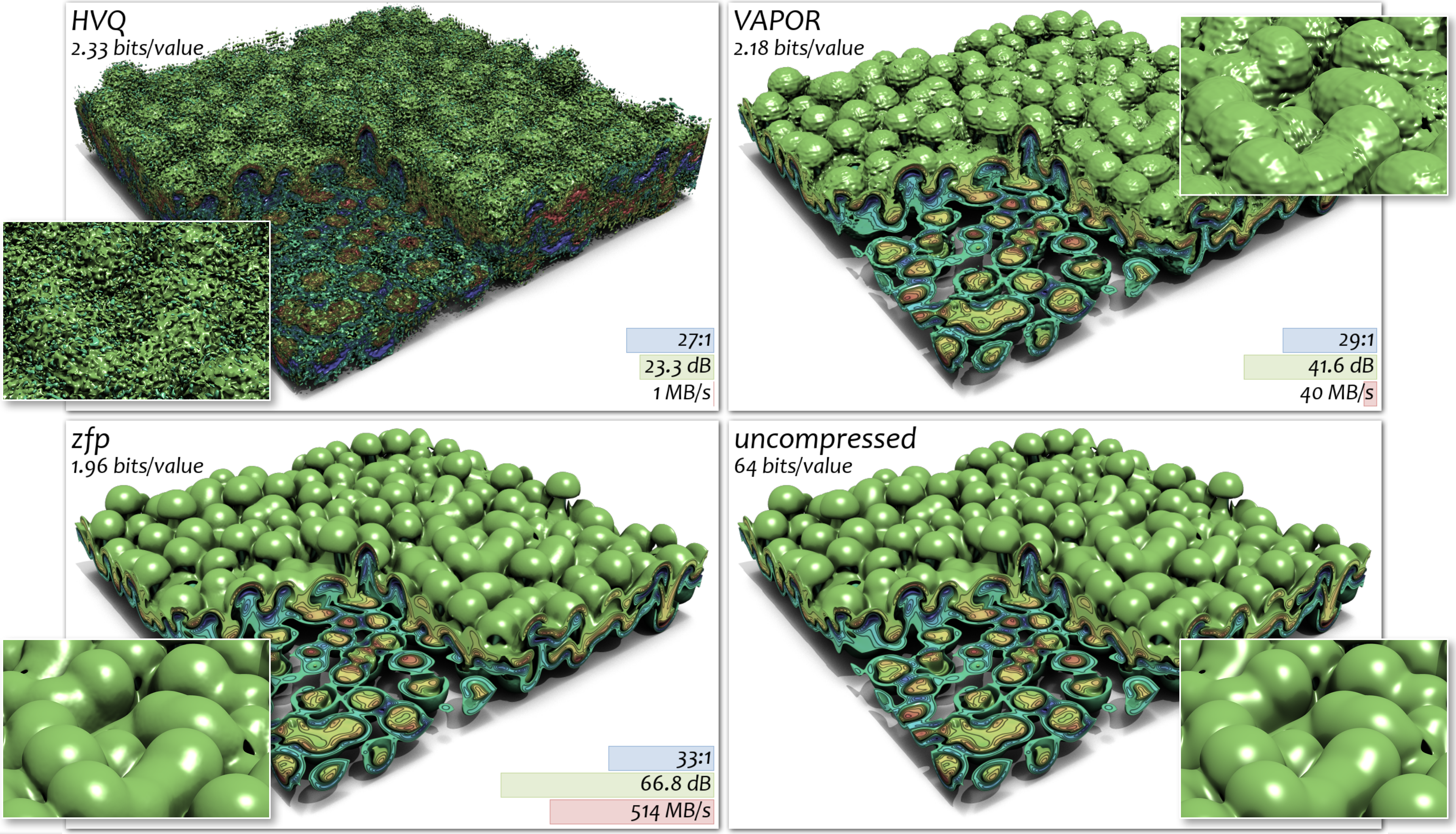
Some compressors yield autocorrelated errors



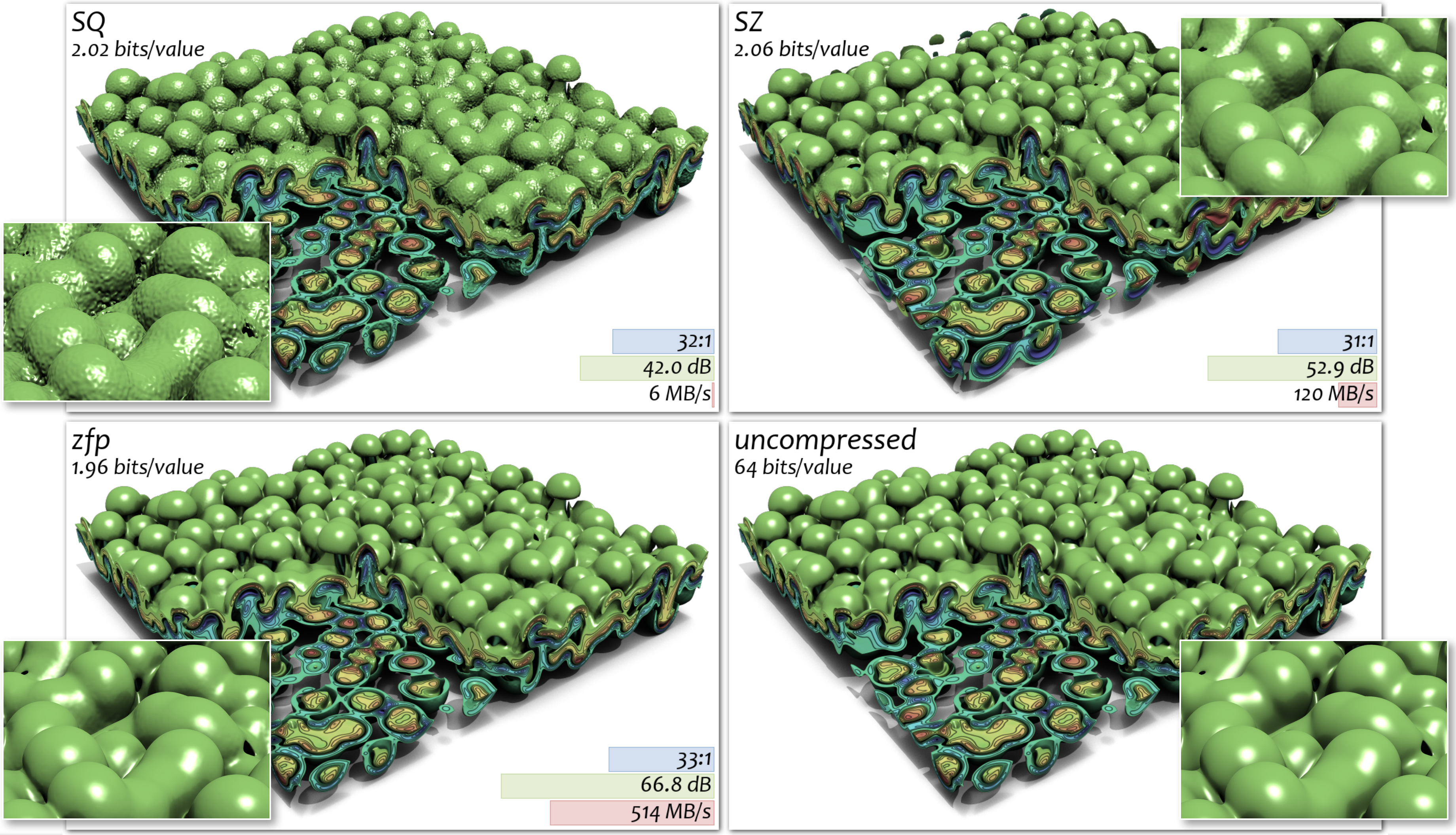
Compressors other than ZFP show artifacts in derivative computations (velocity divergence)



Compressors other than ZFP show artifacts in derivative computations (velocity divergence)



Compressors other than ZFP show artifacts in derivative computations (velocity divergence)



Conclusions

- Lossy data compression can be viable in scientific computing workflows
  - ~100x compression acceptable for visualization
  - ~10x compression acceptable for quantitative data analysis
  - ~4x compression of simulation state with <0.1% error in final quantity of interest
- Little effort has focused on metrics for evaluating compression errors
  - Error distributions can vary greatly between compressors but are rarely considered
    - Difficult to prescribe desired shape of error distribution
  - *Z-checker* tool, developed by Cappello and others at Argonne, is a good first step
- HPC community needs to provide analysis code with simulation results
  - How else can we quantify impact of lossy compression?
  - Need collection of "standard" data sets for evaluating & comparing compressors
- What statistical metrics and properties should we be concerned with?

References

[fpzip] Lindstrom & Isenburg, “Fast and efficient compression of floating-point data,” 2006

[HVQ] Schneider & Westermann, “Compression domain volume rendering,” 2003

[ISABELA] Lakshminarasimhan et al., “ISABELA for effective in situ compression of scientific data,” 2013

[JPEG2000] Woodring et al., “Revisiting wavelet compression for large-scale climate data using JPEG 2000 and ensuring data precision,” 2011

[LP] Ibarria et al., “Out-of-core compression and decompression of large n-dimensional scalar fields,” 2003

[LZ4A, LZ4P] Kunkel et al., “Decoupling the selection of compression algorithms from quality constraints with SCIL,” 2017

[SQ] Iverson et al., “Fast and effective lossy compression algorithms for scientific datasets,” 2012

[SZ] Di & Cappello, “Fast error-bounded lossy HPC data compression with SZ,” 2016

[Tucker] Ballester & Pajarola, “Lossy volume compression using Tucker truncation and thresholding,” 2016

[VAPOR] Clyne et al., “Interactive desktop analysis of high resolution simulations,” 2007

[ZFP] Lindstrom, “Fixed-rate compressed floating-point arrays,” 2014

Lawrence Livermore
National Laboratory