Embedded Zerotree Wavelet (EZW)

These Notes are Based on (or use material from):

1. J. M. Shapiro, “Embedded Image Coding Using Zerotrees of Wavelet Coefficients,” *IEEE Trans. on Signal Processing*, Vol. 41, No. 12, pp. 3445 – 3462, Dec. 1993.
2. J. S. Walker and T. Q. Nguyen, “Wavelet-Based Image Compression,” Ch. 6 in *The Transform and Data Compression Handbook*, edited by K. R. Rao and P. C. Yip, CRC Press, 2001.
3. C. Christopoulos, A. Skodras, and T. Ebrahimi, “The JPEG2000 Still Image Coding System: An Overview,” *IEEE Trans. Cons. Elect.*, Vol. 46., No. 4, pp. 1103 – 1127, Nov. 2000.

Motivation for EZW [1]

- Transform Coding Needs "**Significance Map**" to be sent:
  - At low bit rates a large # of the transform coefficients are quantized to zero ➔ **Insignificant Coefficients**
  - We'd like to not have to actually send any bits to code these
    - That is… allocate zero bits to the insignificant coefficients
  - But… you need to somehow inform the decoder about which coefficients are insignificant
    - JPEG does this using run-length coding: (Run Length, Next Nonzero)
  - In general… Send a **significance map**

Quantized Coefficients

64	56	48	32	24	16	0	0
56	48	40	24	16	23	0	0
40	40	30	24	16	8	0	8
32	32	32	24	24	16	0	0
24	24	16	8	0	0	8	0
16	16	8	0	0	8	0	0
0	0	0	8	0	0	0	0
0	0	0	0	0	0	0	0

Significance Map

1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	1
1	1	1	1	1	1	0	0
1	1	1	1	0	0	1	0
1	1	1	0	0	1	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0

Motivation for EZW (cont.)

Here is a two-stage wavelet decomposition of an image. Notice the large number of zeros (black) but that run-length coding is not likely to be the best approach:

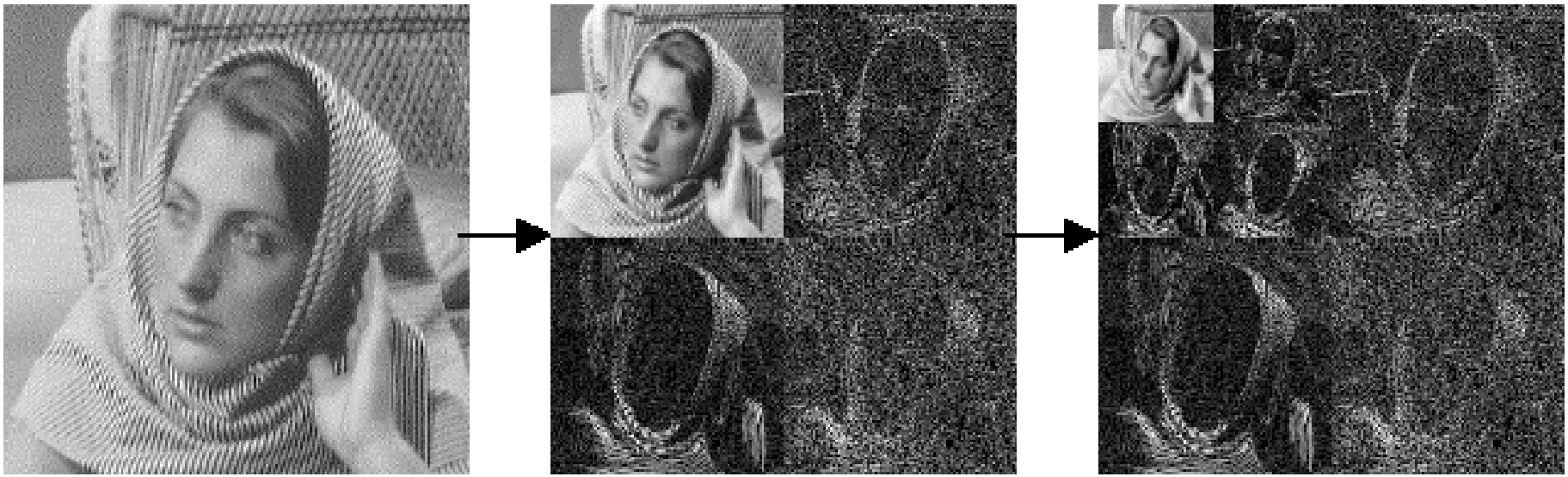


Fig. 5. Example of dyadic decomposition into subbands for the test image 'barbara'

(Figure from [3])

Motivation for EZW (cont.)

But… Couldn't we use entropy coding to make this more efficient? Yes… in fact, that *is* what JPEG does. But it is easy to see that even with entropy coding the significance map (SM) idea get "expensive" as we go to low bit rates [1].

Total Bit Cost = Bit Cost of SM + Bit Cost of Nonzero Values

Under some simplifying conditions (see [1]) Shapiro argued using entropy calculations that the percentage of total cost taken up by SM coding increases as the bit rate decreases!!!

➔SM approach gets increasingly inefficient as we try to compress more!!!!! The cost of specifying where the few significant coefficients are gets large at low rates

Example (see [1]): at 0.5 bit/pixel must use 54% of bits for SM

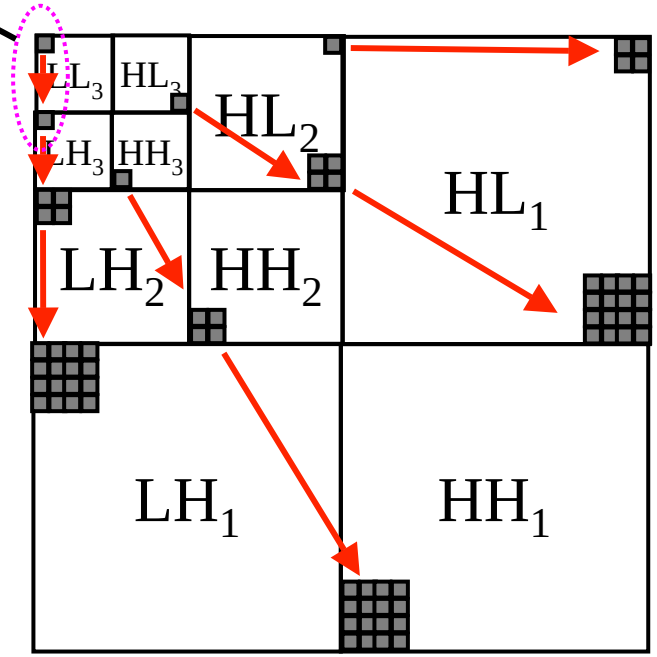
Shapiro's Wavelet Idea for Solving SM Problem

Quad Trees:

> Same Spatial Region

> Different Resolution Levels (sub-bands)

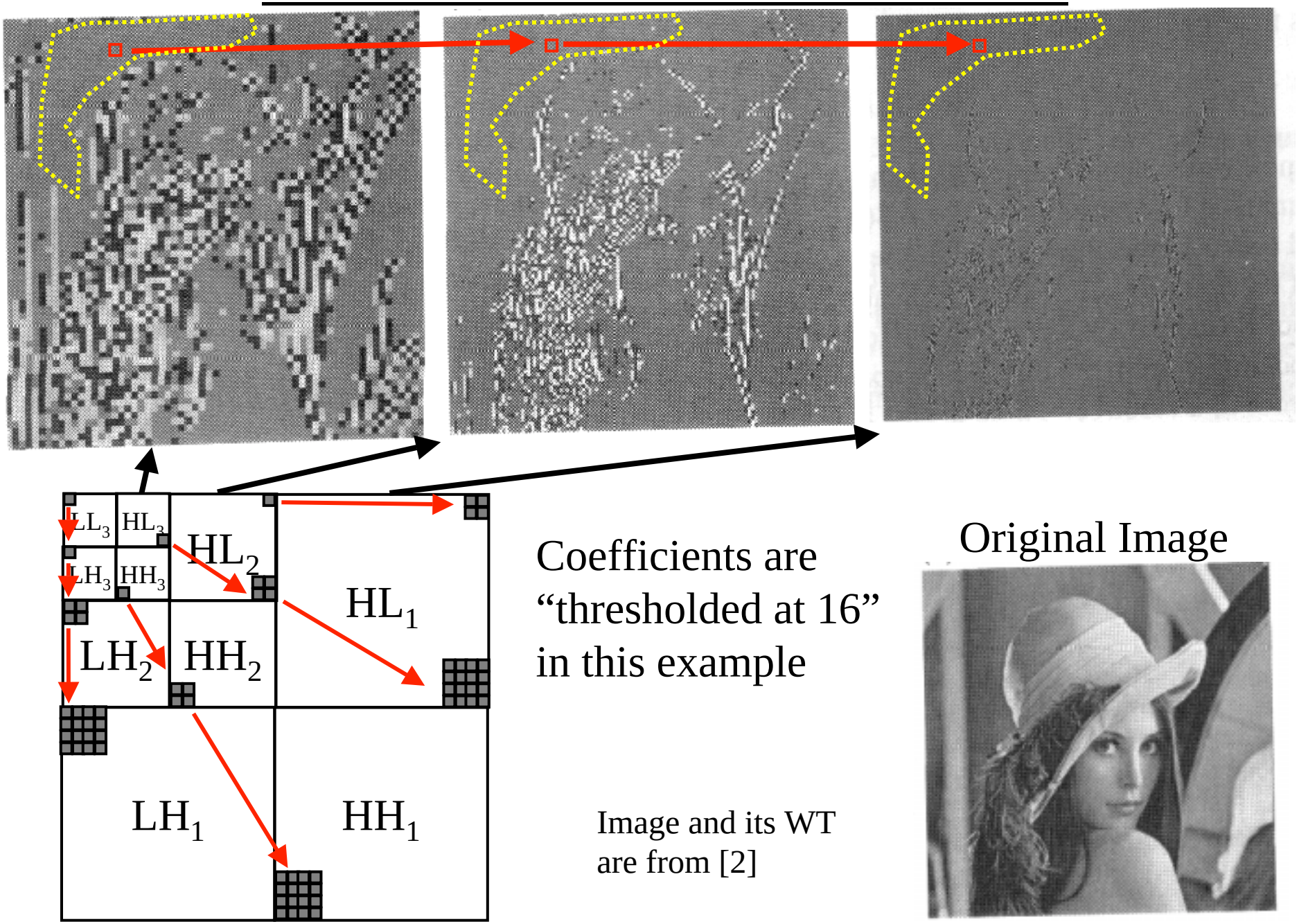
For final, LL subband: one covers the same area as one pixel in the "detail" space above



Idea: An insignificant coefficient is VERY likely to have all of its "descendents" on its quad tree also be insignificant

> Such a coefficient is called a "**Zerotree Root**"

Illustration of Zerotree Occurance



Coefficients are "thresholded at 16" in this example

Image and its WT are from [2]

Original Image

How Do Zerotrees Help?

The previous chart showed the prevalence of zerotrees. Now… how do they help with the SM problem?

You only have to tell the decoder where a zero root lies… it can figure out where all the descendent zeros on the tree lie by using the rule for generating quadtrees.

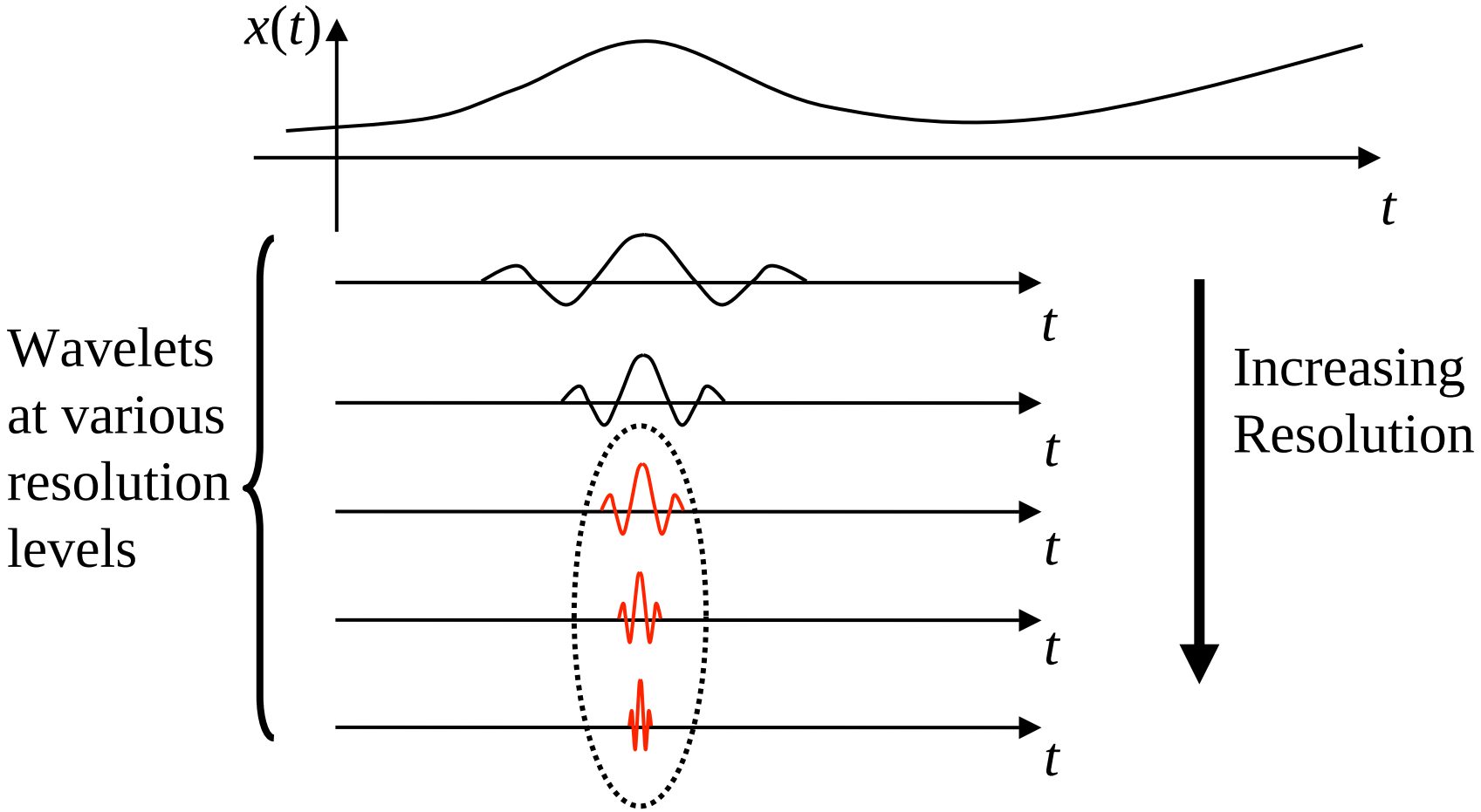
So… there is an agreed upon a rule and it so happens that zerotree roots happen alot when trying to code at low bit rates…

At low bit rates, zerotree roots occur frequently even at the coarse subband levels and that leads to long trees… and that very efficiently conveys the SM info

Note: we don't really rely on a true SM, but we convey it using a model

What Causes Zerotrees?

Use 1-D example to illustrate:



These wavelets have insignificant inner product with signal at this location

Successive Approximation: The Other Part of EZW

While zerotrees are a major part of EZW they are not the only significant part…

The other part has to do with embedded coding.

The goal of embedded coding is to create a bit stream that can be truncated at any point by the decoder….. AND you get a reconstructed signal that is R-D optimal for the number of bits so far received!

There are many ways to do this. EZW uses a successive approximation view of quantization…
… and it links this idea to zerotree coding in a way that allows zerotrees to be highly exploited.

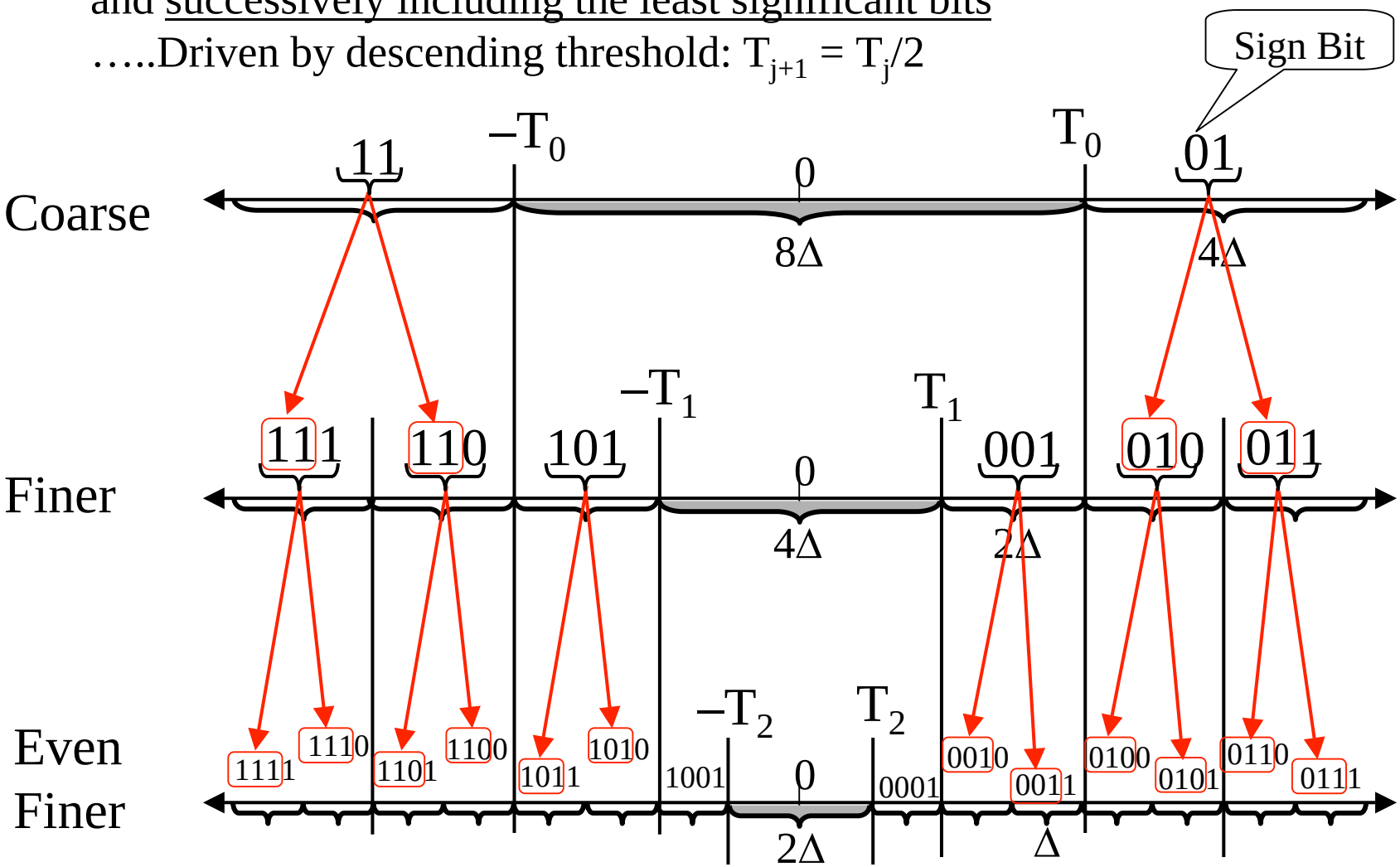
Successive Approximation Quantizer

Start with coarsest and successively refine to the finest…

…. equivalent to starting with most significant magnitude bit
(sign bit is handled separately)

and successively including the least significant bits

…..Driven by descending threshold: $T_{j+1} = T_j/2$



Successive Approximation Quantizer (cont.)

Applying the SA quantizer with in EZW:

- Compute the wavelet transform of the image
- Set a threshold T_0 near the middle of the range of WT coefficient magnitudes
- This gives a large "dead zone" that creates of lots of "insignificant values"
  - These give rise to lots of zerotrees
  - Zerotrees efficiently handle significance map problem
  - Send MSB's of significant coefficients
- Then reduce threshold: $T_{j+1} = T_j/2$
  - This causes some former insig coeff to become significant
  - ➔ only have to tell where new significance has occurred
  - For previously significant: refine by sending next finer bit

EZW Algorithm

Sequence of Decreasing Thresholds: $T_o, T_1, \ldots, T_{N-1}$

with $T_i = T_{i-1}/2$ and $|\text{coefficients}| < 2\,T_o$

Maintain Two Separate Lists:

- Dominant List: *coordinates* of coeffs not yet found significant
- Subordinate List: *magnitudes* of coefficients already found to be significant

For each threshold, perform two passes: Dominant Pass then Subordinate Pass

Dominant Pass (Significance Map Pass)

- Coeff's on Dominant List (i.e. currently insig.) are compared to T_i
  - asking: has this coeff become significant at the new threshold?
- The resulting significance map is zero-tree coded and sent:
  - Code significance using four symbols:
    - Zerotree Root (ZTR)
    - Isolated Zero (IZ)
    - Positive Significant (POS)
    - Negative Significant (NEG)
  - For each coeff that has now become significant (POS or NEG)
    - put its magnitude on the Subordinate List (making it eligible for future refinement)
    - remove it from the Dominant List (because it has now been found significant)

Entropy Code using an Adaptive AC

EZW Algorithm (cont.)

Subordinate Pass (Significance Coefficient Refinement Pass)

- Provide next lower signif. bit on the magnitude of each coeff on Subord List
  - Halve the quantizer cells to get the next finer quantizer
  - If magnitude of coeff is in upper half of old cell, provide "1"
  - If magnitude of coeff is in lower half of old cell, provide "0"
- Entropy code sequence of refinement bits using an adaptive AC

Now repeat with next lower threshold

- Stop when total bit budget is exhausted

- Encoded stream is an embedded stream
  - At first you get an "optimal" low rate version
  - As more bits come you get a successively better distortion
  - Can terminate at any time prior to reaching the "full-rate" version