Old and new coding techniques for data compression, error correction and nonstandard situations

Lecture I: data compression … data encoding

Efficient information encoding to:
- reduce **storage** requirements,
- reduce **transmission** requirements,
- often reduce **access time** for storage and transmission

(decompression is usually faster than HDD, e.g. 500 vs 100MB/s)

Jarosław Duda
Nokia, Kraków
25. IV. 2016

Symbol frequencies from a version of the British National Corpus containing 67 million words (http://www.yorku.ca/mack/uist2011.html#f1):

Brute: $\boldsymbol{lg}(\mathbf{27}) \approx \mathbf{4.75}$ **bits/symbol** $(x \to 27x + s)$

Huffman uses: $H' = \sum_i p_i \boldsymbol{r_i} \approx \mathbf{4.12}$ bits/symbol

Shannon: $H = \sum_i p_i \mathbf{lg}(\mathbf{1}/\boldsymbol{p_i}) \approx \mathbf{4.08}$ bits/symbol

We can theoretically improve by $\approx$ **1% here**

$\boldsymbol{\Delta H} \approx \mathbf{0.04}$ **bits/symbol**

Order 1 Markov: ~ **3.3** bits/symbol (~gzip)

Order 2: ~**3.1** bps, word: ~**2.1** bps (~ZSTD)

Currently best compression: cmix-v9 (PAQ)

(http://mattmahoney.net/dc/text.html)

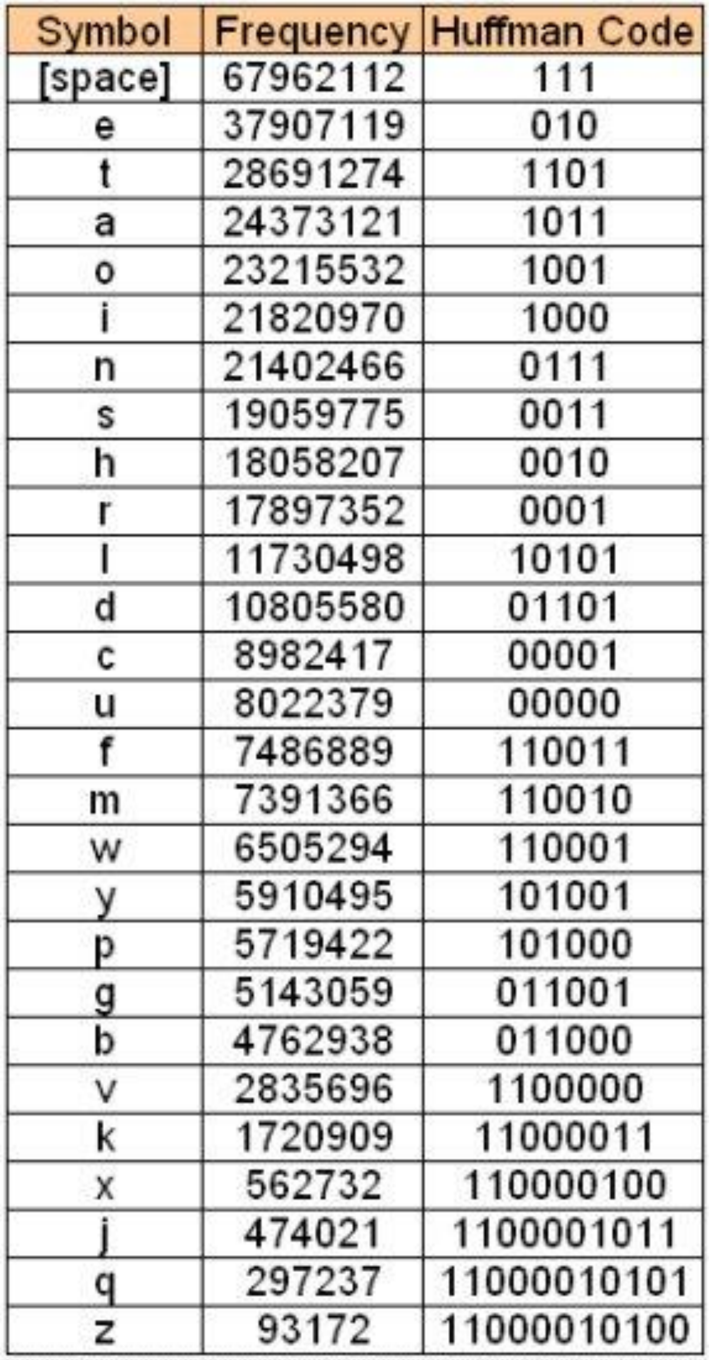
10^9 bytes of text from Wikipedia (enwik9)

into **123**874398 bytes: $\approx$ **1** bit/symbol

$10^8 \to$ **156**27636 $10^6 \to$ **181**476

Hilberg conjecture: $H(n) \sim n^\beta, \quad \beta < 0.9$

... lossy video compression: > 100× reduction



Symbol	Frequency	Huffman Code
[space]	67962112	111
e	37907119	010
t	28691274	1101
a	24373121	1011
o	23215532	1001
i	21820970	1000
n	21402466	0111
s	19059775	0011
h	18058207	0010
r	17897352	0001
l	11730498	10101
d	10805580	01101
c	8982417	00001
u	8022379	00000
f	7486889	110011
m	7391366	110010
w	6505294	110001
y	5910495	101001
p	5719422	101000
g	5143059	011001
b	4762938	011000
v	2835696	1100000
k	1720909	11000011
x	562732	110000100
j	474021	1100001011
q	297237	11000010101
z	93172	11000010100

Lossless compression – fully reversible,

e.g. gzip, gif, png, JPEG-LS, FLAC, lossless mode in h.264, h.265 (~50%)

Lossy – better compression at cost of distortion

Stronger distortion – better compression (rate distortion theory, psychoacoustics)

General purpose (lossless, e.g. gzip, rar, lzma, bzip, zpaq, Brotli, ZSTD) vs **specialized compressors** (e.g. audiovisual, text, DNA, numerical, mesh,…)

For example **audio** (http://www.bobulous.org.uk/misc/audioFormats.html):

Wave: 100%, FLAC: 67%, Monkey's: 63%, MP3: 18%

720p **video** - uncompressed: ~1.5 GB/min, lossy h.265: ~ 10MB/min

Lossless very data dependent, **speed-ratio tradeoff**

PDF: ~80%, RAWimage: 30-80%, binary: ~20-40%, txt: ~12-30%, source code: ~20%, 10000URLs: ~20%, XML:~6%, xls: ~2%, mesh: <1%

General scheme **for Data Compression … data encoding:**

1) Transformations, predictions

To decorrelate data, make it more predictable, encode only new information

Delta coding, YCrCb, Fourier, Lempel-Ziv, Burrows-Wheeler, MTF, RLC …

2) If lossy, quantize coefficients (e.g. $c \to \mathrm{round}(c/Q)$)

---

sequence of ↓ **(context_ID, symbol)**

3) Statistical modeling of final events/symbols

Static, parametrized, stored in headers, context-dependent, adaptive

4) (Entropy) coding: use $\lg(1/p_s)$ bits, $H = \sum_s p_s \lg(1/p_s)$ bits total

Prefix/Huffman: fast/cheap, but inaccurate ($p \sim 2^{-r}$)

Range/arithmetic: costly (multiplication), but accurate

Asymmetric Numeral Systems – fast and accurate

general (Apple LZFSE, Fb ZSTD), DNA (CRAM), games (LZNA, BitKnit), Google VP10, WebP

Part 1: Transformations, predictions, quantization

Delta coding: write relative values (differences) instead of absolute

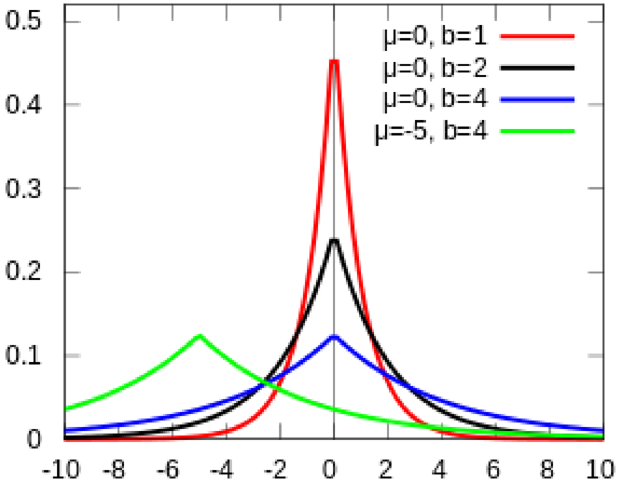
e.g. time $(1, 5, 21, 28, 34, 44, 60) \rightarrow (1, 4, 16, 7, 6, 10, 6)$

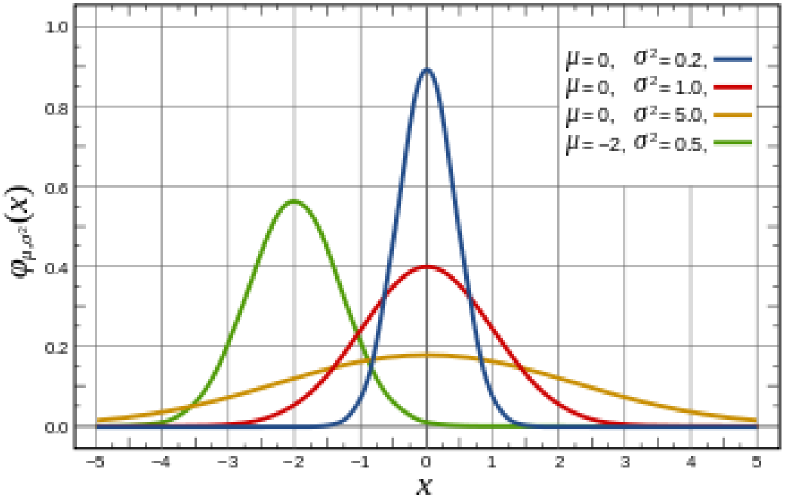
the differences often have some **parametric probability distribution**:

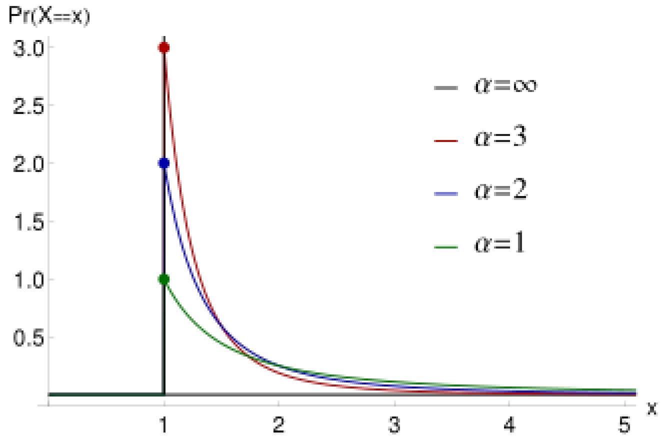
Laplacian distribution (geometric): $\Pr(x) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$

Gaussian distribution (normal): $\Pr(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

Pareto distribution: $\Pr(x) \propto \frac{1}{x^{\alpha+1}}$

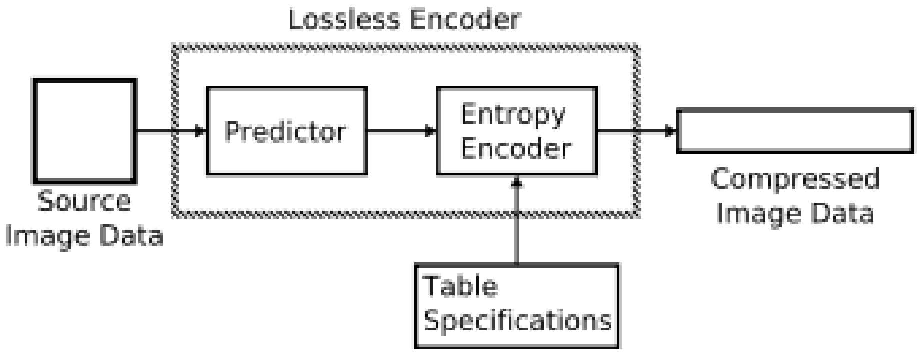






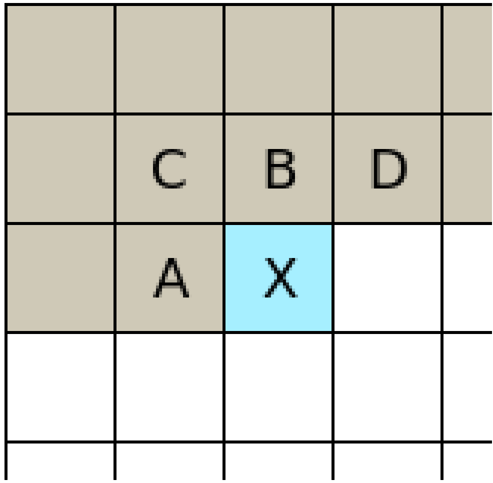
JPEG LS – simple prediction

https://en.wikipedia.org/wiki/Lossless_JPEG



- scan pixels line-by-line
- **predict** X (edge detection):

$$X = \begin{cases} \min(A,B) & \text{if } C \geq \max(A,B) \\ \max(A,B) & \text{if } C \leq \min(A,B) \\ A+B-C & \text{otherwise.} \end{cases}$$



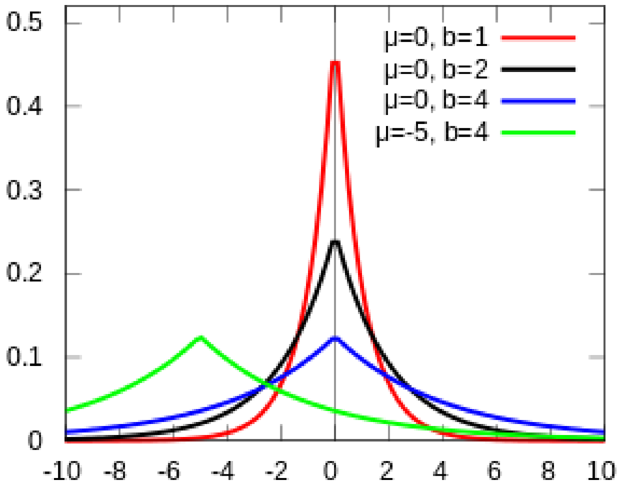
- find **context** determining coding parameters:

  $(\boldsymbol{D-B},\ \boldsymbol{B-C},\ \boldsymbol{C-A})$ $\left(\frac{9^3+1}{2} = 365 \text{ contexts}\right)$

- (entropy) **coding** of **differences** using Golomb codes

(good prefix code for **Laplace distribution**)

with **parameter *m* accordingly to context**

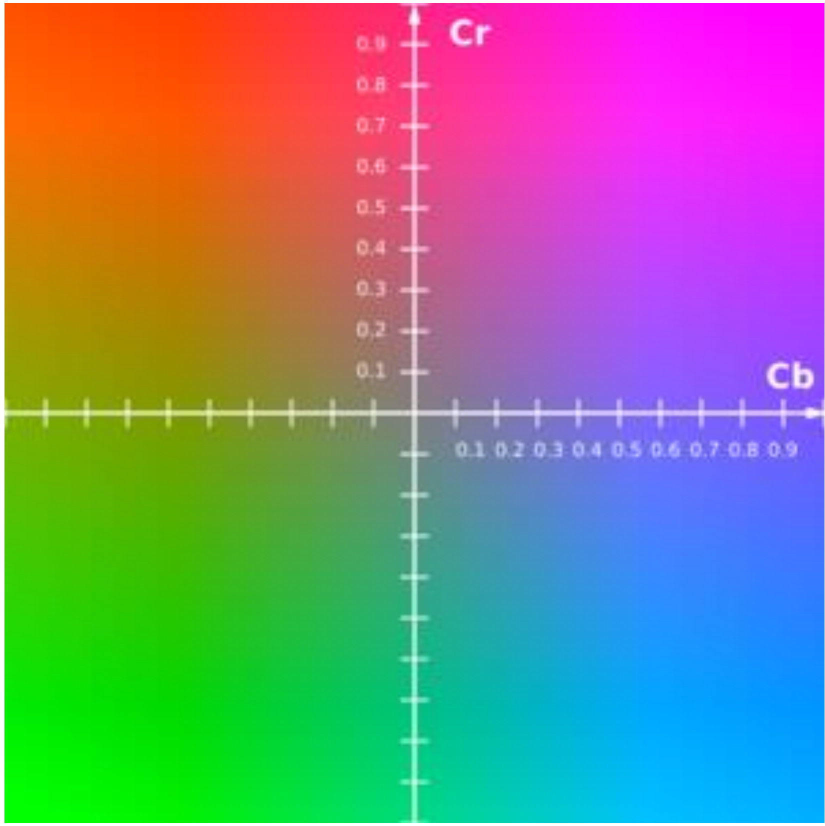


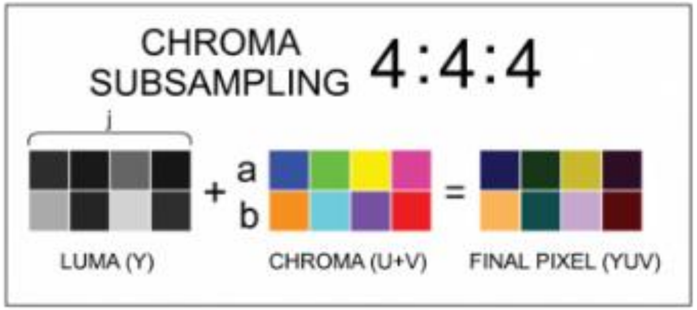
Color transformation to better exploit **spatial correlations**

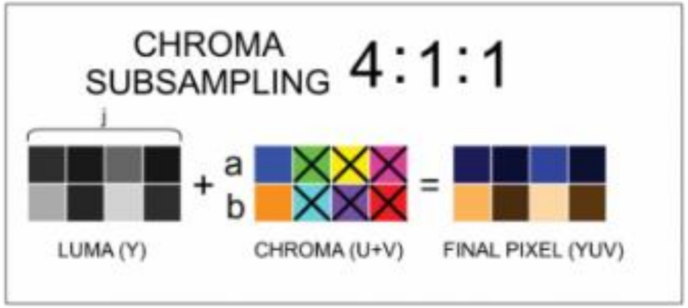
YCbCr:

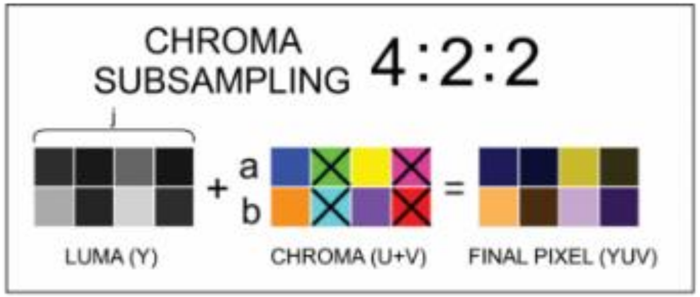
Y – luminance/brightness

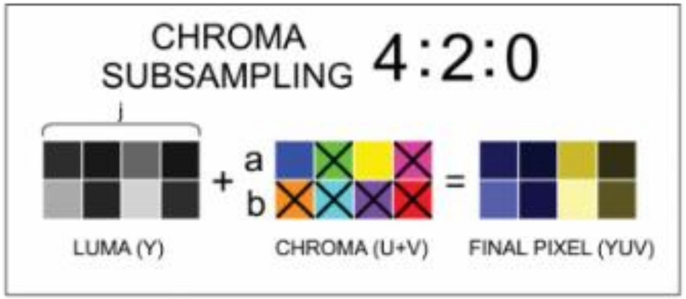
Cb, Cr – blue, red difference











Chroma subsampling
In **lossy** compression
(usually 4:2:0)

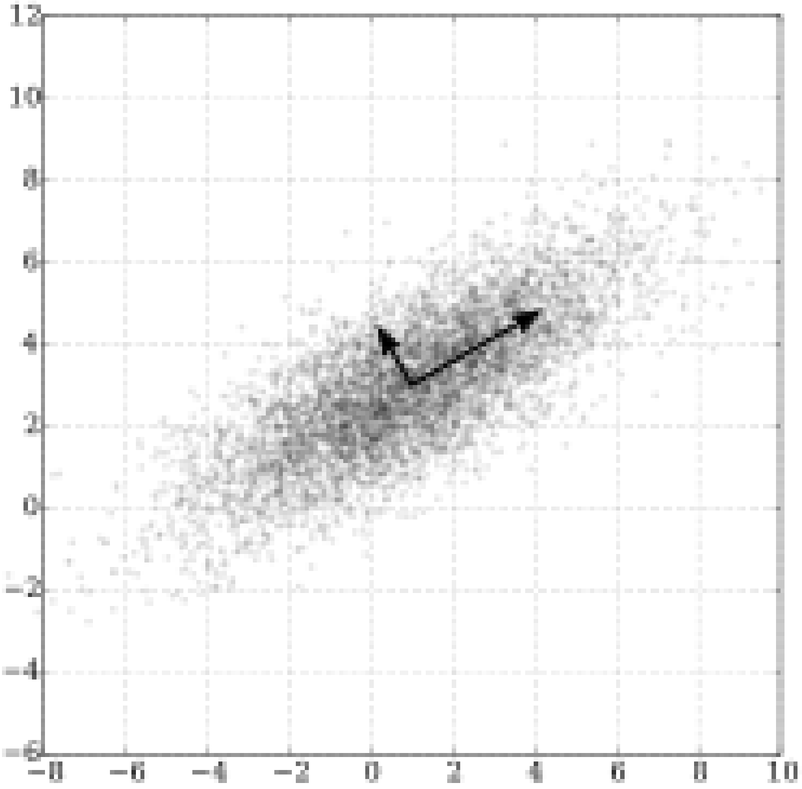
Color depth:
3x8 – true color
3x10,12,16 – deep color

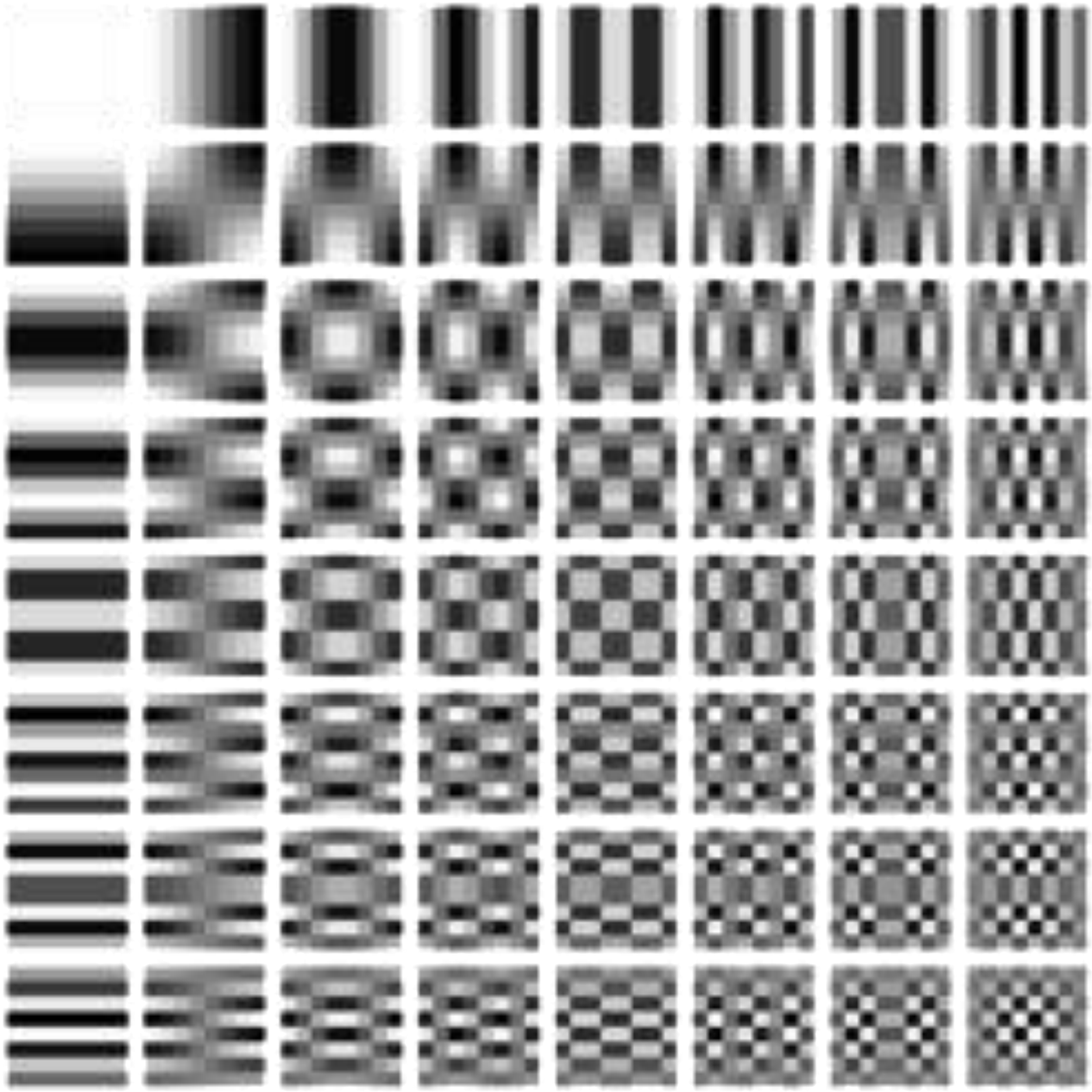
Karhunen–Loève transform to **decorrelate** into **independent variables**

Often close to Fourier (DCT):

(energy preserving, frequency domain)

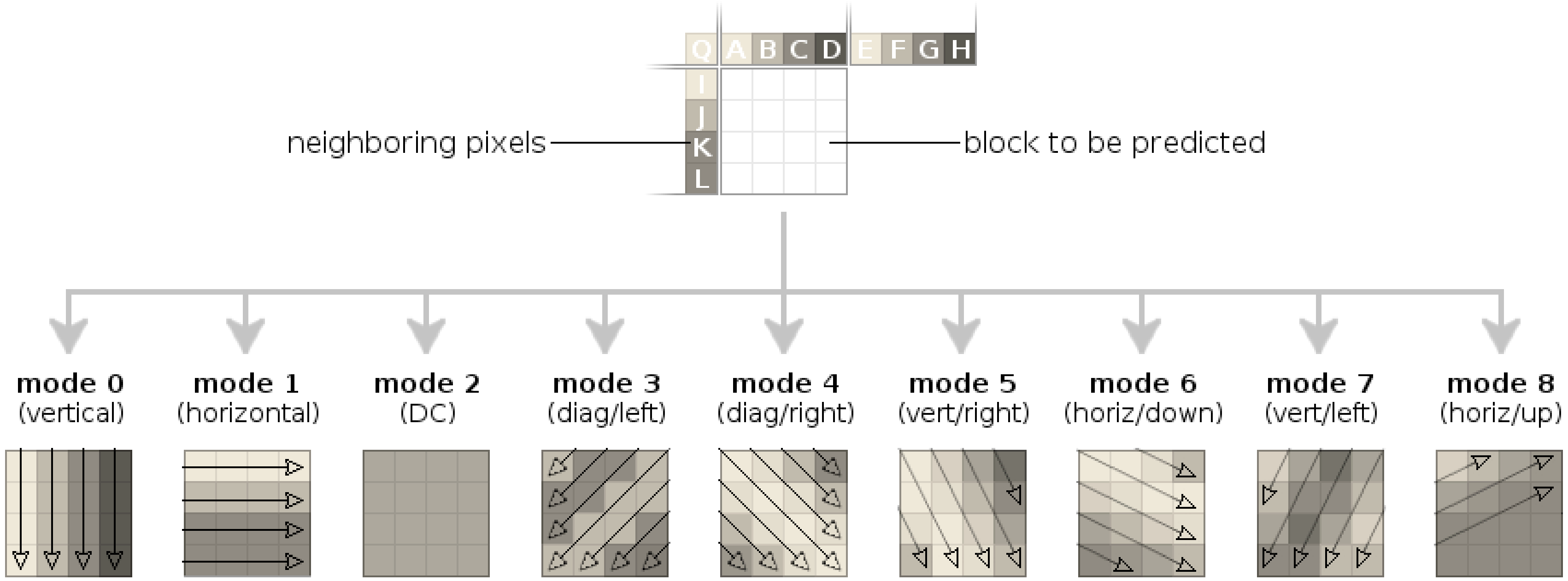
(stored quantized in JPEG)





Principal Component Analysis

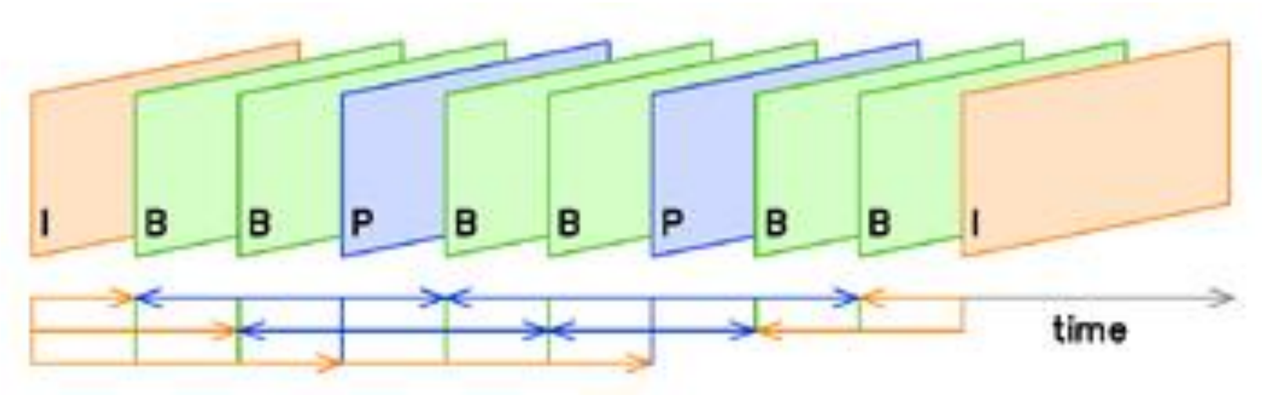
Video: predict values (in blocks), encode only difference (residue)



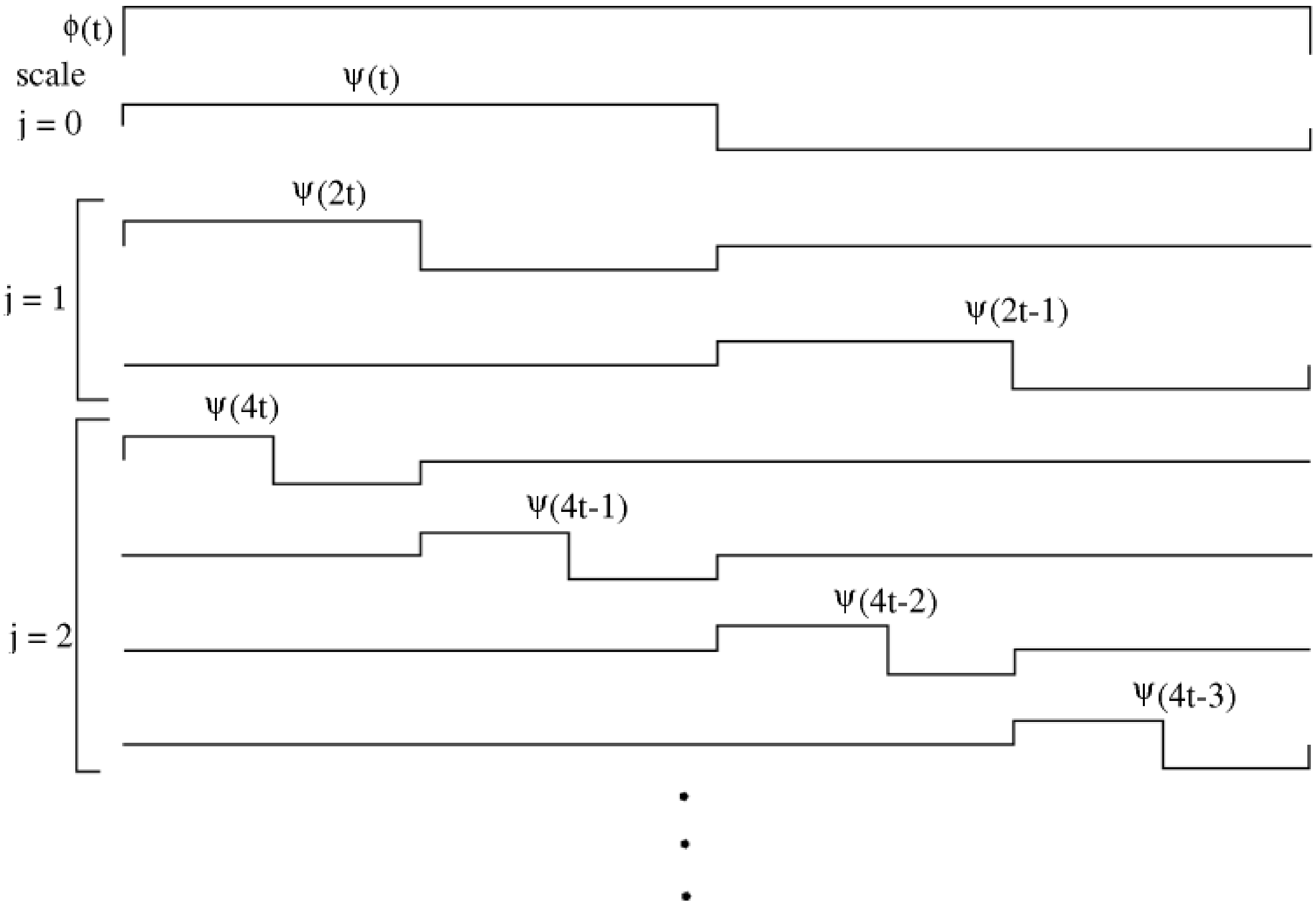
AVC/H.264 intra prediction modes

intra-prediction (in a frame, choose and encode mode)**,**

inter-prediction (between frames, use **motion** of objects)

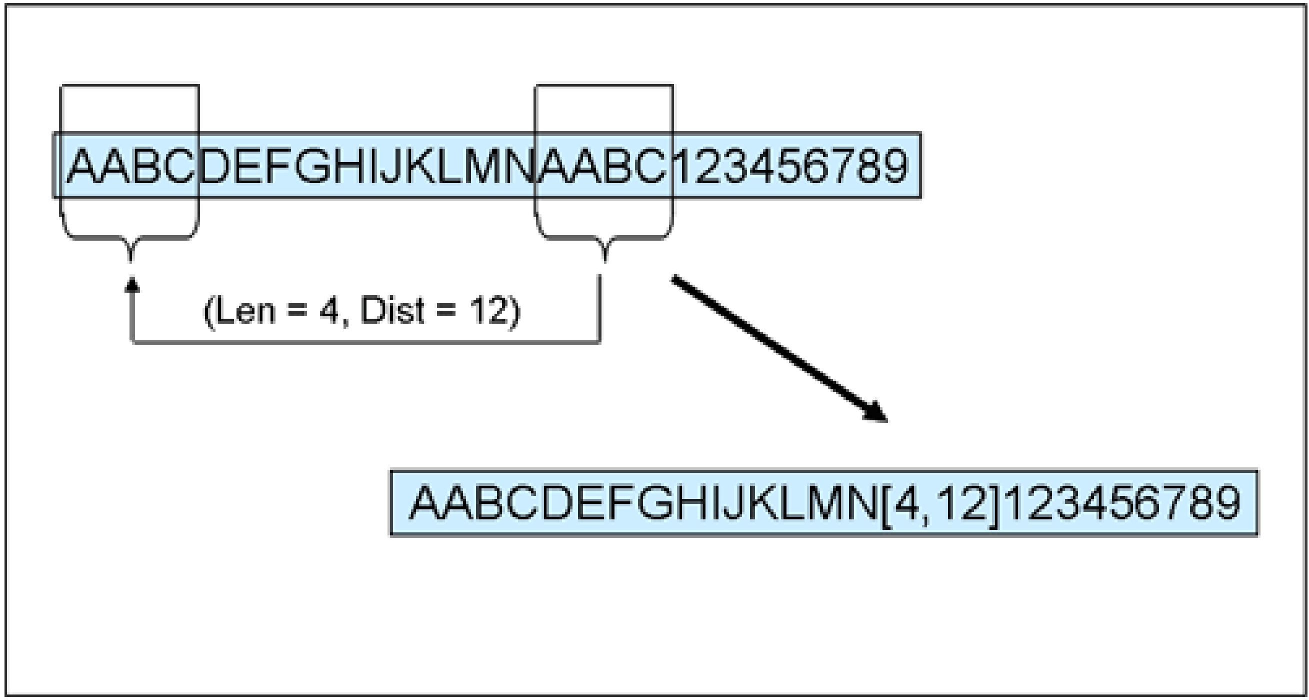


DCT – fixed support, Wavelets – varying support, e.g. JPEG2000
Haar wavelets:



Progressive decoding: send low frequencies first, then further to improve quality

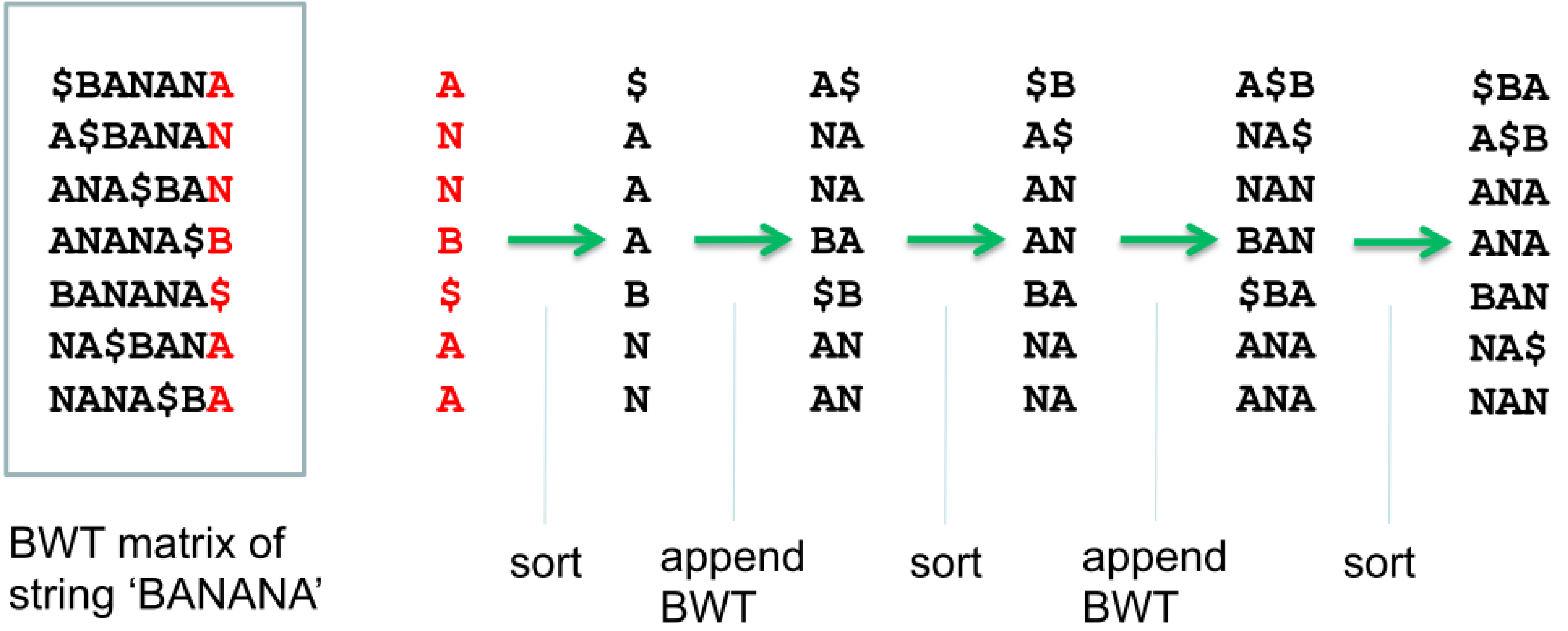
Lempel-Ziv (large family)
Replace repeats with position and length
(decoding is just parsing)



https://github.com/inikep/lzbench
i5-4300U,
100MB total files (from W8.1):

	method	Compr.[MB/s]	Decomp.[MB/s]	Ratio
lz4fast r131 level 17	LZ	994	3172	74 %
lz4 r131	LZ	497	2492	62 %
lz5hc v1.4.1 level 15	LZ	2.29	724	44 %
Zlib (gzip) level 1	LZ + Huf	45	197	49 %
Zlib (gzip) level 9	LZ + Huf	7.5	216	45 %
ZStd v0.6.0 level 1	LZ + tANS	231	595	49 %
ZStd v0.6.0 level 22	LZ + tANS	2.25	446	37 %
Google Brotli level 0	LZ + o1 Huf	210	195	50 %
Google Brotli level 11	LZ + o1 Huf	0.25	170	36 %

Burrows – Wheeler Transform: sort lexicographically all cyclic shifts, take last column



$BANANA	A	$	A$	$B	A$B	$BA
A$BANAN	N	A	NA	A$	NA$	A$B
ANA$BAN	N	A	NA	AN	NAN	ANA
ANANA$B	B	A	BA	AN	BAN	ANA
BANANA$	$	B	$B	BA	$BA	BAN
NA$BANA	A	N	AN	NA	ANA	NA$
NANA$BA	A	N	AN	NA	ANA	NAN

BWT matrix of string 'BANANA'

sort

append BWT

sort

append BWT

sort

Usually improves statistical properties for compression (e.g. text, DNA)

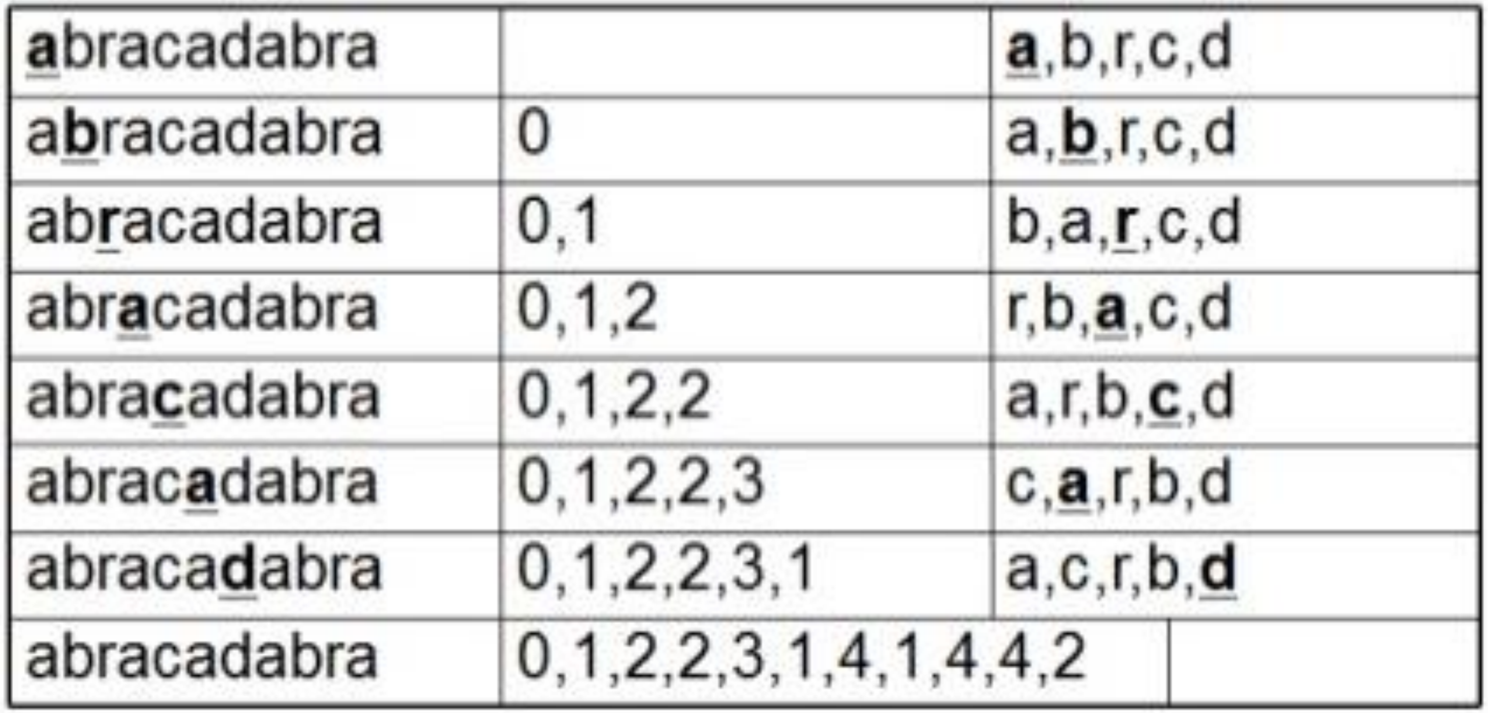
https://sites.google.com/site/powturbo/entropy-coder benchmark:

	enwik9 (1GB)	enwik9 + BWT	Speed (BWT, MB/s)
O1 rANS	55.7 %	21.8 %	193/403
TurboANX (tANS)	63.5 %	24.3 %	492/874
Zlibh (Huffman)	63.8 %	28.5 %	307/336

Run-Length Encoding (RLE) – encode repeat lengths

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWW
WWWWWWWWWBWWWWWWWWWWWWWW → **12W1B12W3B24W1B14W**

Move-to-front (MTF, Bentley 86) – move last symbol to front of alphabet



abracadabra		**a**,b,r,c,d
a**b**racadabra	0	a,**b**,r,c,d
ab**r**acadabra	0,1	b,a,**r**,c,d
abr**a**cadabra	0,1,2	r,b,**a**,c,d
abra**c**adabra	0,1,2,2	a,r,b,**c**,d
abrac**a**dabra	0,1,2,2,3	c,**a**,r,b,d
abraca**d**abra	0,1,2,2,3,1	a,c,r,b,**d**
abracadabra	0,1,2,2,3,1,4,1,4,4,2	

Often used with BWT:

mississipi → (BWT) → pssmipissii → (MTF) → 23033313010

“To be or not to be…” : 7807 bits of entropy (symbol-wise)

7033 bits after MTF, 6187 bits after BWT+MTF (7807 after BWT only)

Binning – most significant bits of floats often have characteristic distribution

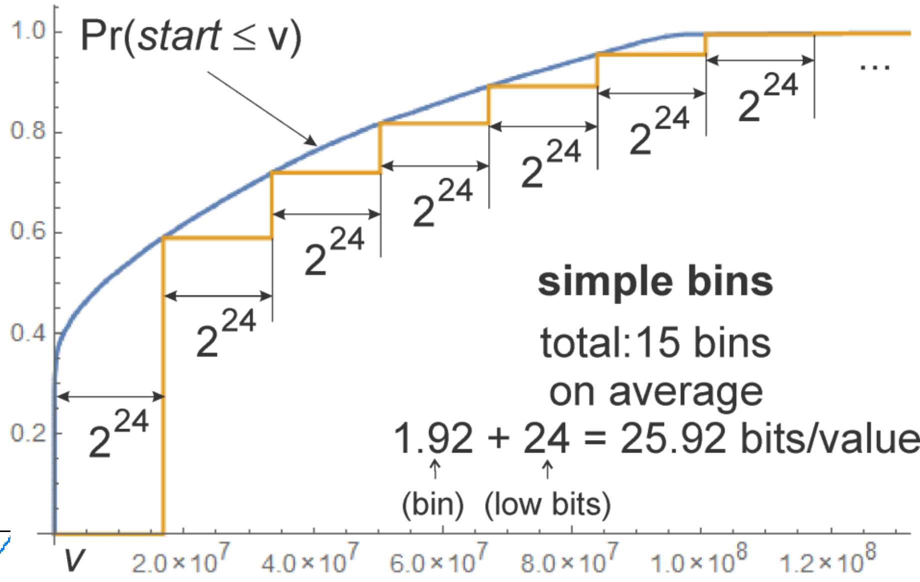
Quantization, e.g. $c \rightarrow \mathrm{round}(c/Q)$

store only bin number

(rate distortion theory)

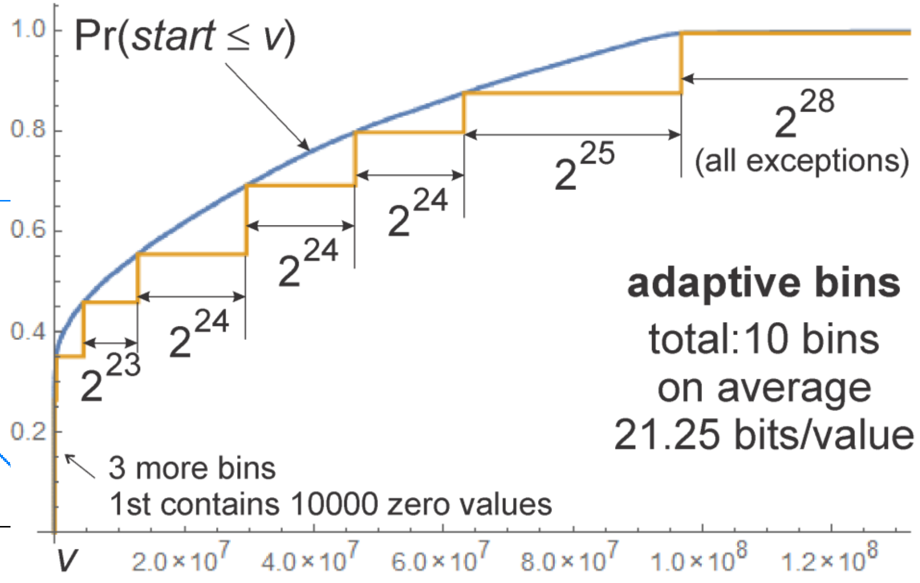
Vector quantization (many coordinates at once)

- (r, angle) for energy conservation (e.g. Daala)
- For more convenient representation



simple bins

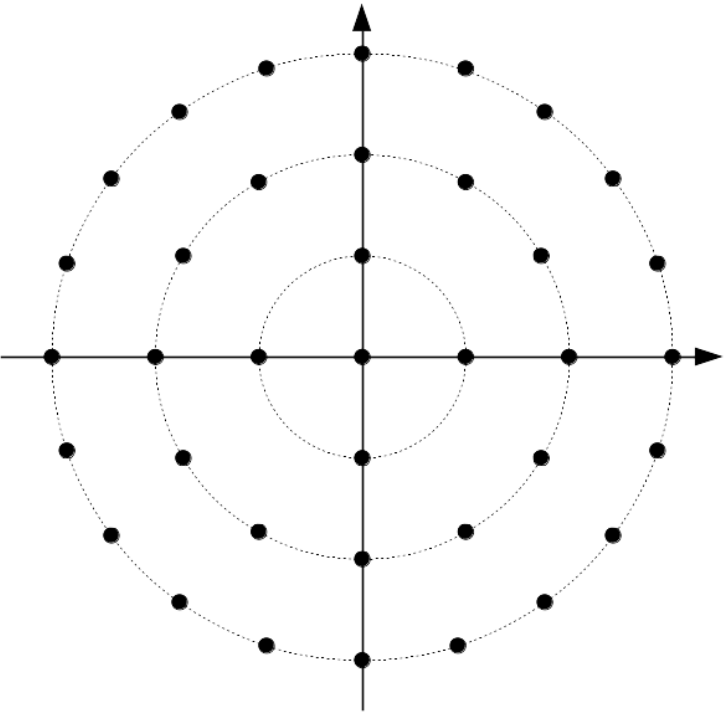
total:15 bins
on average
1.92 + 24 = 25.92 bits/value
(bin) (low bits)

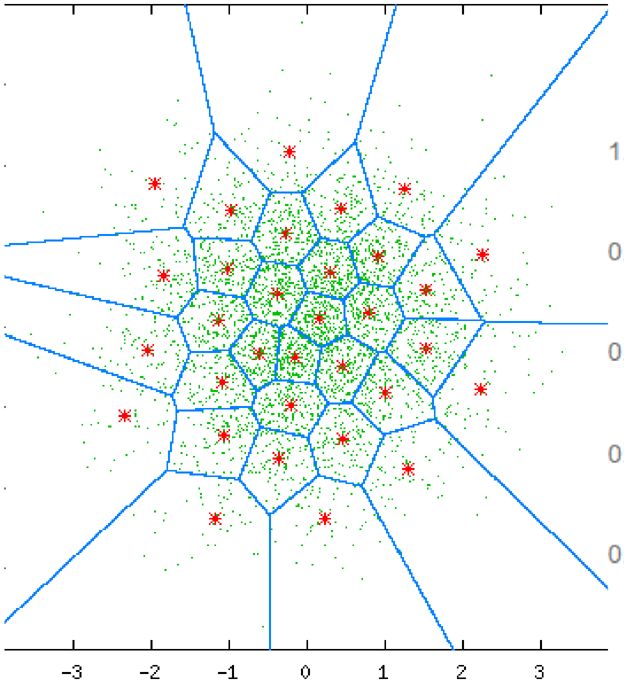


adaptive bins

total:10 bins
on average
21.25 bits/value

3 more bins
1st contains 10000 zero values





Part 2: (Entropy) coding – the heart of data compression

Transformations, predictions, quantization etc.

have lead to a **sequence of symbols/event we finally need to encode**

Optimal storage of symbol of probability $\boldsymbol{p}$ uses $\mathbf{lg}(\mathbf{1}/\boldsymbol{p})$ bits

Using **approximated** probabilities means **suboptimal** compression ratio

We need to **model these probabilities**, then use **(entropy) coder** with them

For example **unary coding:** $\mathbf{0} \rightarrow 0,\ \mathbf{1} \rightarrow 10,\ \mathbf{2} \rightarrow 110,\ \mathbf{3} \rightarrow 1110,\ \mathbf{4} \rightarrow 11110, \ldots$

Is optimal if $\Pr(\mathbf{0}) = 0.5, \Pr(\mathbf{1}) = 0.25, \ldots, \Pr(\boldsymbol{k}) = 2^{-k-1}$ (geometric)

Encoding (p_i) distribution with entropy coder optimal for (q_i) distribution costs

$$\Delta H = \sum_i p_i \lg(1/q_i) - \sum_i p_i \lg(1/p_i) = \sum_i p_i \lg\left(\frac{p_i}{q_i}\right) \approx \frac{1}{\ln(4)} \sum_i \frac{(p_i - q_i)^2}{p_i}$$

more bits/symbol - so called **Kullback-Leibler “distance”** (not symmetric)

Imagine we want to **store n values (e.g. hash) of m bits**: directly $m \cdot n$ bits

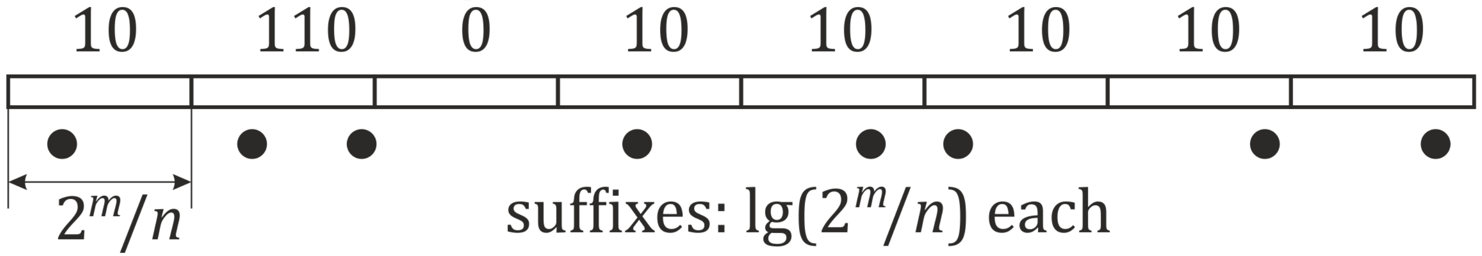
Their **order** contains $\lg(n!) \approx \lg((n/e)^n) = n\lg(n) - n\lg(e)$ bits of information

If order is not important, we could use only $\approx n(m - \lg(n) + 1.443)$

It means **halving storage** for 178k of 32bit hashes, 11G of 64bit hashes

How to cheaply **approximate** it?

overhead: counting elements in buckets - unary code, **2**n bits



Split the range (2^m positions) into n equal size buckets

- Use unary system to write number of hashes in each bucket: $2n$ bits
- Write suffixes inside buckets: $\lg(2^m/n)$ for each hash:

$$2n + n \cdot \lg(2^m/n) = n(m - \lg(n) + 2)$$ bits total

Overhead can be cheaply reduced to $\lg(3) \approx 1.585 = 1.443 + 0.142$ bits

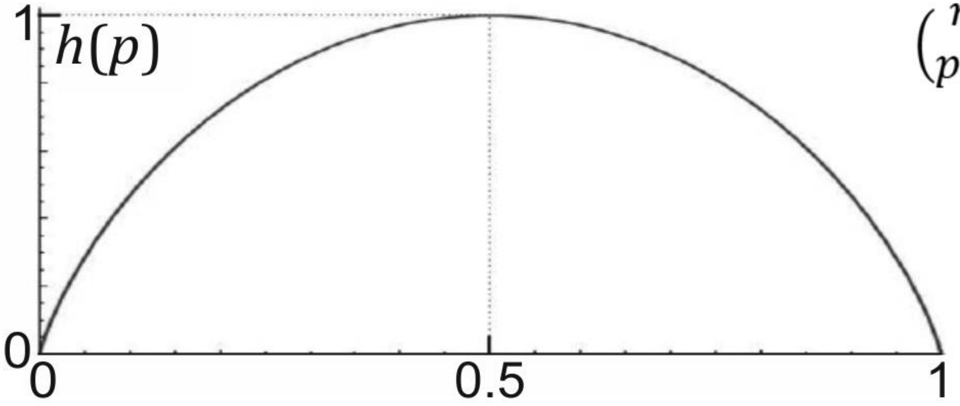
By using 1 **trit** coding instead of unary (James Dow Allen):

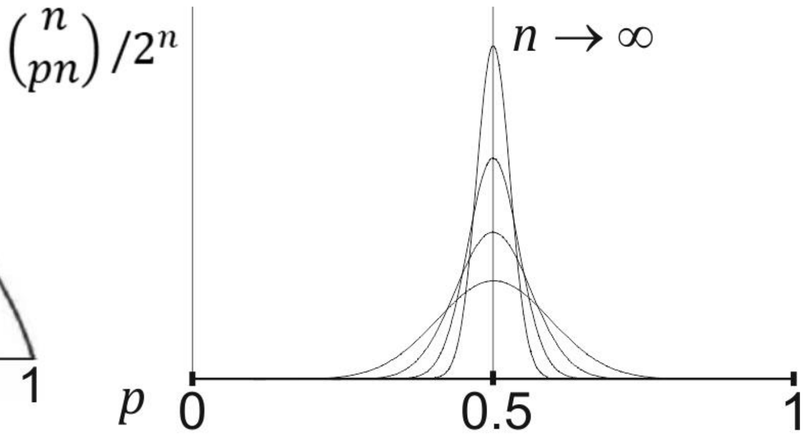
“0”, ”1”, ”2 or more”,

then store prefixes: $a_1 < a_2 <..< a_{k-2} < a_k > a_{k-1}$

We need n bits of information to choose one of 2^n possibilities.

For length n 0/1 sequences with pn of “1”, how many bits we need to choose one?





$$\binom{n}{pn} \approx 2^{nh(p)} \quad \text{for} \quad h(p) := -p\lg(p) - (1-p)\lg(1-p)$$

$$\frac{n!}{(pn)!\,((1-p)n)!} \approx \frac{(n/e)^n}{(pn/e)^{pn}\big((1-p)n/e\big)^{(1-p)n}} = 2^{n\lg(n) - pn\lg(pn) - (1-p)n\lg\left((1-p)n\right)} = 2^{n(-p\lg(p) - (1-p)\lg(1-p))}$$

Entropy coder: encodes sequence with $(p_s)_{s=0..m-1}$ probability distribution using asymptotically at least $\boldsymbol{H = \sum_s p_s \lg(1/p_s)}$ **bits/symbol** $(H \le \lg(m))$

Seen as weighted average: **symbol of probability p contains $\lg(1/p)$ bits.**

Statistical modelling – where the probabilities come from

- **Frequency counts** – e.g. for each block (~140B/30kB) and written in header,
- Assume a **parametric distribution**, estimate its parameter – for example

$$\Pr(x) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right), \quad \Pr(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad \Pr(x) \propto \frac{1}{x^{\alpha+1}}$$

- **Context dependence**, like neighboring pixels, similar can be grouped
  e.g. order 1 Markov (context is the previous symbol): use $\Pr(s_i|s_{i-1})$,
- **Prediction by partial matching** (PPM) – varying number of previous symbols as context (build and update suffix tree),
- **Context mixing** (CM, ultracompressors like PAQ) – combine predictions from multiple contexts (e.g. PPM) using some machine learning.

Static or **adaptive** – modify probabilities (dependent on decoded symbols), modify CDF toward CDF of recently processed data block

Enwik9: CM 12.4%, PPM 15.7%, BWT 16%, dict 16.5%, LZ+bin 19.3%, LZ+byt: 21.5%, LZ: 32%

Decoding: 100h cmix, 1h mons, 400s M03 , 15s glza , 55s LZMA , 2.2s ZSTD , 3.7s lz5

Adaptiveness – modify probabilities (dependent on decoded symbols), e.g.

```
for (int i = 1; i < m; i++) CDF[i] -= (CDF[i] - mixCDF[i]) >> rate;
```

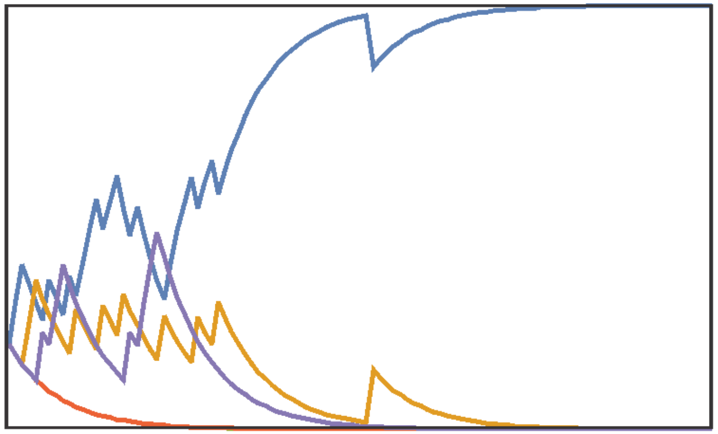
where `CDF[i]` = $\sum_{j<i} \Pr(j)$, `mixCDF[]` is CDF for recent data block

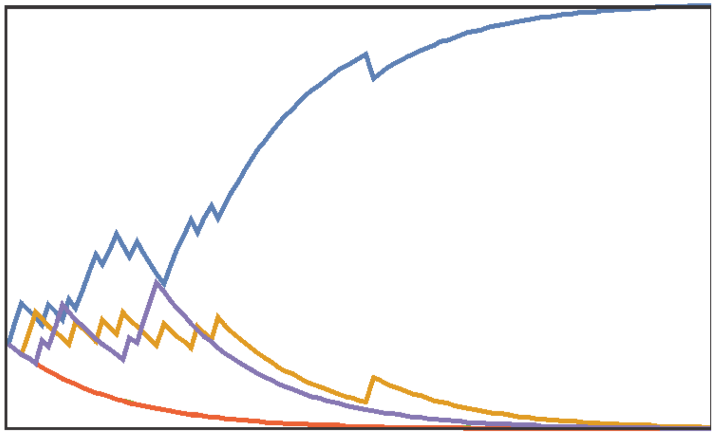
example: {0014044010001001404441000010010000000000000000000000
1000000000000000000000000000000000000000000000000000}

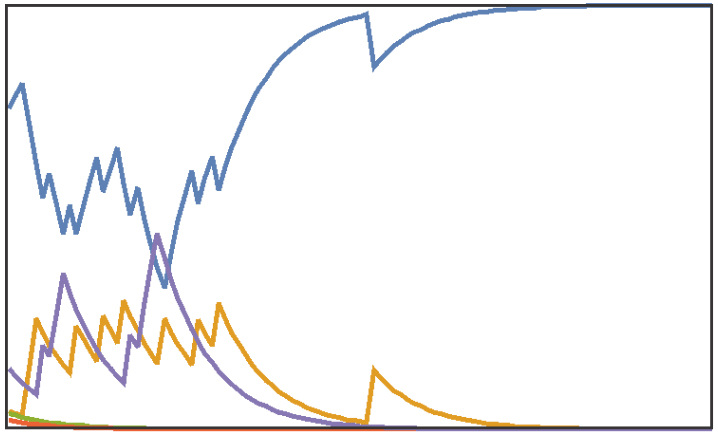
{0,1,2,3,4} alphabet, length = 104, **direct encoding:** $104 \cdot \lg(5) \approx$ **241** bits

Static probability (stored?) : {89, 8, 0, 0, 7}/104 $104 \cdot \sum_i p_i \lg(1/p_i) \approx$ **77** bits

Symbol-wise adaptation starting from flat (uniform) or averaged **initial distribution**:


flat, rate = 3, $-\sum \lg(\Pr) \approx$ **70**


flat, rate = 4, $-\sum \lg(\Pr) \approx$ **78**


avg, rate = 3, $-\sum \lg(\Pr) \approx$ **67**

Some **universal codings,** upper bound doesn't need to be known

unary coding: 0 → 0, **1** → 10, **2** → 110, **3** → 1110, **4** → 11110, ...

Is optimal if $\Pr(\mathbf{0}) = 0.5, \Pr(\mathbf{1}) = 0.25, \dots,$ $\Pr(\boldsymbol{x}) = 2^{-x-1}$ (**geometric**)

Golomb coding: choose parameter M (preferably a power of 2).

Write $\lfloor \boldsymbol{x}/\boldsymbol{M} \rfloor$ with **unary** coding, then **directly $\boldsymbol{x}$ mod $\boldsymbol{M}$**

Needs $\lfloor x/M \rfloor + 1 + \lg(M)$ bits,

optimal for ~**geometric** distribution: $\Pr(x) \approx \frac{2}{M} \cdot 2^{-\left\lfloor \frac{x}{M} \right\rfloor} \approx \left(\sqrt[M]{2}\right)^{-x}$

Elias gamma coding: write $N = \lfloor \lg(x) \rfloor$ zeroes, then $N+1$ bits of x

Uses $2\lfloor \lg(x) \rfloor + 1$ bits – optimal for **Pareto**: $\Pr(x) \approx 2^{-2\lfloor \lg(x) \rfloor + 1} \approx x^{-2}$

Directly writing x costs $\lfloor \lg(x) \rfloor + 1$ bits → $\Pr(x) \approx \frac{1}{x}$, but **length** also costs

Elias delta coding – $\lg(\lg(x))$ with unary, optimal for $\Pr(x) \approx \frac{1}{x \lg^2(x)}$

Elias omega coding – recursive, optimal for $\Pr(x) \approx \frac{1}{x \cdot \lg(x) \cdot \lg(\lg(x)) \cdot \dots}$

symbol of probability p
contains $\lg(1/p)$ bits of information

symbol → length r bit sequence
assumes: $\Pr(\text{symbol}) \approx 2^{-r}$

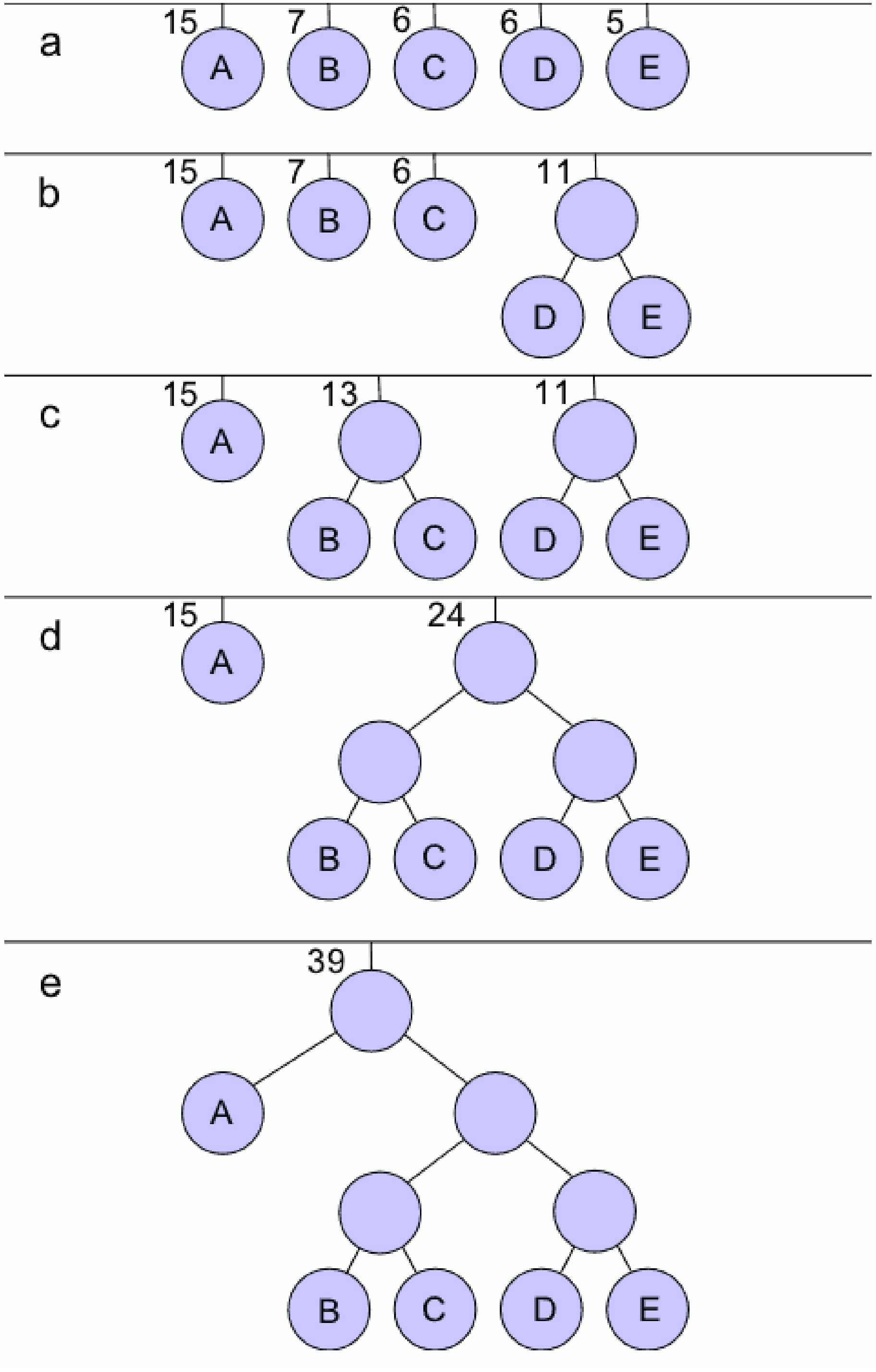
Huffman coding: perfect prefix code

- Sort symbols by frequencies
- Replace two least frequent with tree of them and so on

generally: **prefix codes**
defined by a tree
unique decoding:
no sequence is a prefix of another

http://en.wikipedia.org/wiki/Huffman_coding

$A \rightarrow 0, B \rightarrow 100, C \rightarrow 101, D \rightarrow 110, E \rightarrow 111$



Encoding (p_i) distribution with entropy coder optimal for (q_i) distribution costs

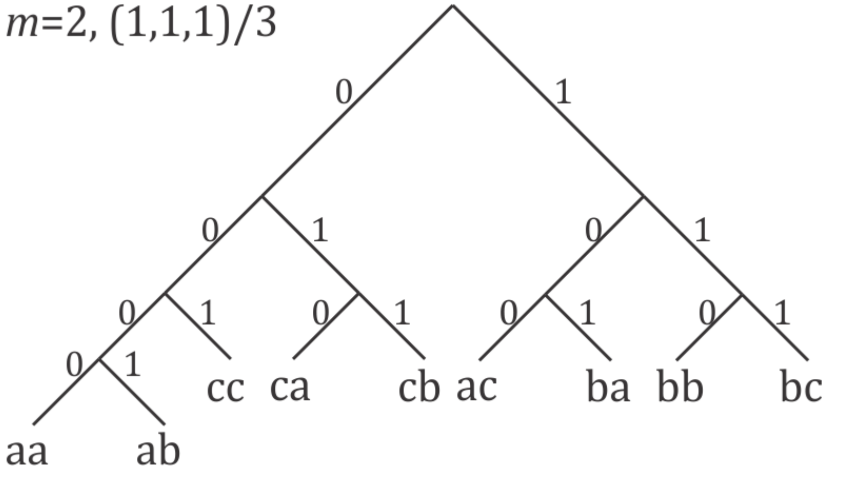
$$\Delta H = \sum_i p_i \lg(1/q_i) - \sum_i p_i \lg(1/p_i) = \sum_i p_i \lg\left(\frac{p_i}{q_i}\right) \approx \frac{1}{\ln(4)} \sum_i \frac{(p_i - q_i)^2}{p_i}$$

more bits/symbol - so called **Kullback-Leibler "distance"** (not symmetric).

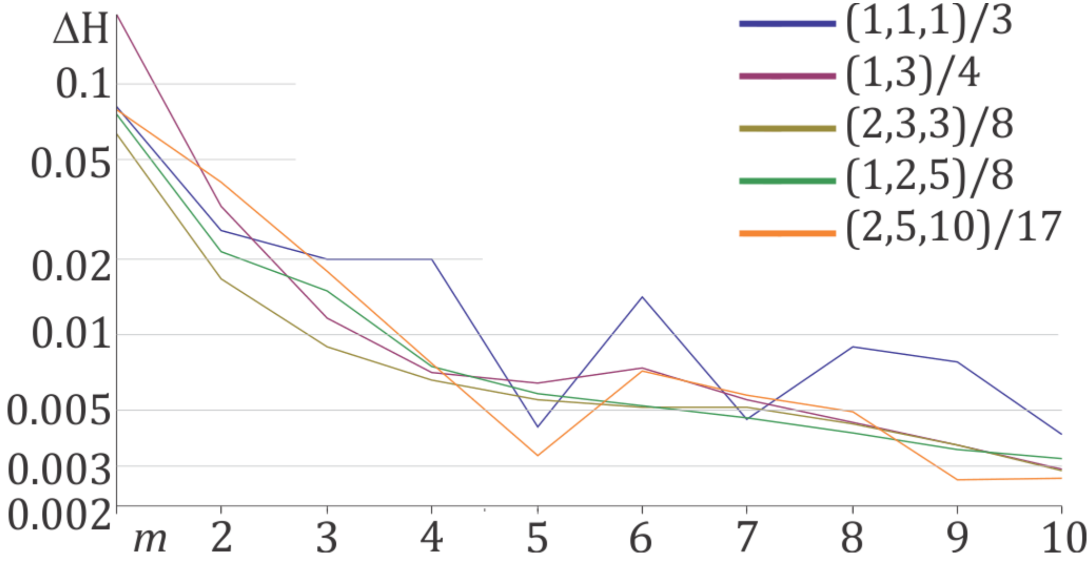
Huffman inaccurate, terrible for skewed distributions (Pr(a) >> 0.5)

e.g. Pr(a) = 0.99, Pr(b) = 0.01, H ~ 0.08 bits/symbol, Huffman: $a \rightarrow 0$, $b \rightarrow 1$

We can reduce ΔH by **grouping** m symbols together (alphabet size 2^m or 3^m):

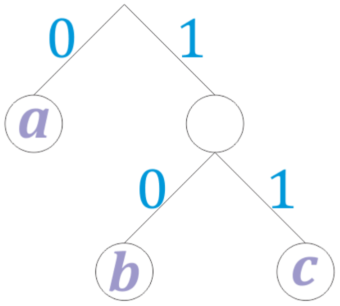


ΔH=(2·4 + 7·3)/9/2 - lg(3) ≈ 0.026 bits/symbol



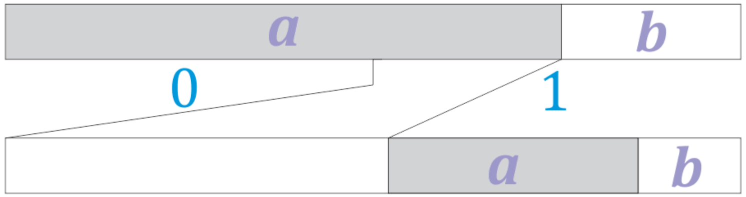
$\Delta H \propto \sim 1/m$ but cost grows like 2^m

Past: compromise



Huffman coding
(also unary, Golomb, Elias, etc.)
fast (>300MB/s/core)
no multiplication, needs sorting
but **inaccurate:** Pr(*a*) ~ 2^{-r}
e.g. uses 1 bit/symbol
for Pr(*a*)=0.01, Pr(*b*)=0.99

or ?



arithemtic/range coding
slow (<< 100MB/s/core)
uses multiplication
uses nearly **accurate** Pr(*a*)
e.g. uses ~0.08 bits/symbol
for Pr(*a*)=0.01, Pr(*b*)=0.99

Now: ANS

Asymmetric Numeral Systems (ANS)
fast (> 500MB/s/core)
no multiplication or sorting (uses state $X \in \mathbb{N}$)
uses nearly **accurate** Pr(*a*)
e.g. uses ~0.08 bits/symbol
for Pr(*a*)=0.01, Pr(*b*)=0.99
allows for simultaneus encryption

tANS decoding:

$X \to s$, new X

$\mathbf{0} \to a , 2 + d_1$

$\mathbf{1} \to b , 0 + 2d_2 + d_1$

$\mathbf{2} \to a , 0$

$\mathbf{3} \to a , 1$

newX *nbBits*

decodingTable

Huffman vs **ANS** in compressors (LZ + entropy coder):

from Matt Mahoney benchmarks http://mattmahoney.net/dc/text.html

	Enwiki8 100,000,000B	**encode time [ns/byte]**	**decode time [ns/byte]**
ZSTD 0.6.0 –22 --ultra	25,405,601	701	2.2
Brotli (Google Feb 2016) –q11 w24	25,764,698	3400	5.9
LZA 0.82b –mx9 –b7 –h7	26,396,613	449	9.7
lzturbo 1.2 –39 –b24	26,915,461	582	2.8
WinRAR 5.00 –ma5 –m5	27,835,431	1004	31
WinRAR 5.00 –ma5 –m2	29,758,785	228	30
lzturbo 1.2 –32	30,979,376	19	2.7
zhuff 0.97 –c2	34,907,478	24	3.5
gzip 1.3.5 –9	36,445,248	101	17
pkzip 2.0.4 –ex	36,556,552	171	50
WinRAR 5.00 –ma5 –m1	40,565,268	54	31
ZSTD 0.4.2 –1	40,799,603	7.1	3.6

zhuff, **ZSTD** (Yann Collet, Facebook): LZ4 + tANS (switched from Huffman)

lzturbo (Hamid Bouzidi): LZ + tANS (switched from Huffman)

LZA (Nania Francesco): LZ + rANS (switched from range coding)

saving time and energy in extremely frequent task

Apple LZFSE =
Lempel-Ziv + Finite State Entropy
Default in iOS9 and OS X 10.11

"matching the compression ratio of ZLIB level 5, but with much higher energy efficiency and speed (between 2x and 3x) for both encode and decode operation"

Finite State Entropy is
Yann Collet's (known from e.g. LZ4)
implementation of tANS

Default DNA compression: CRAM 3.0 of European Bioinformatics Institute

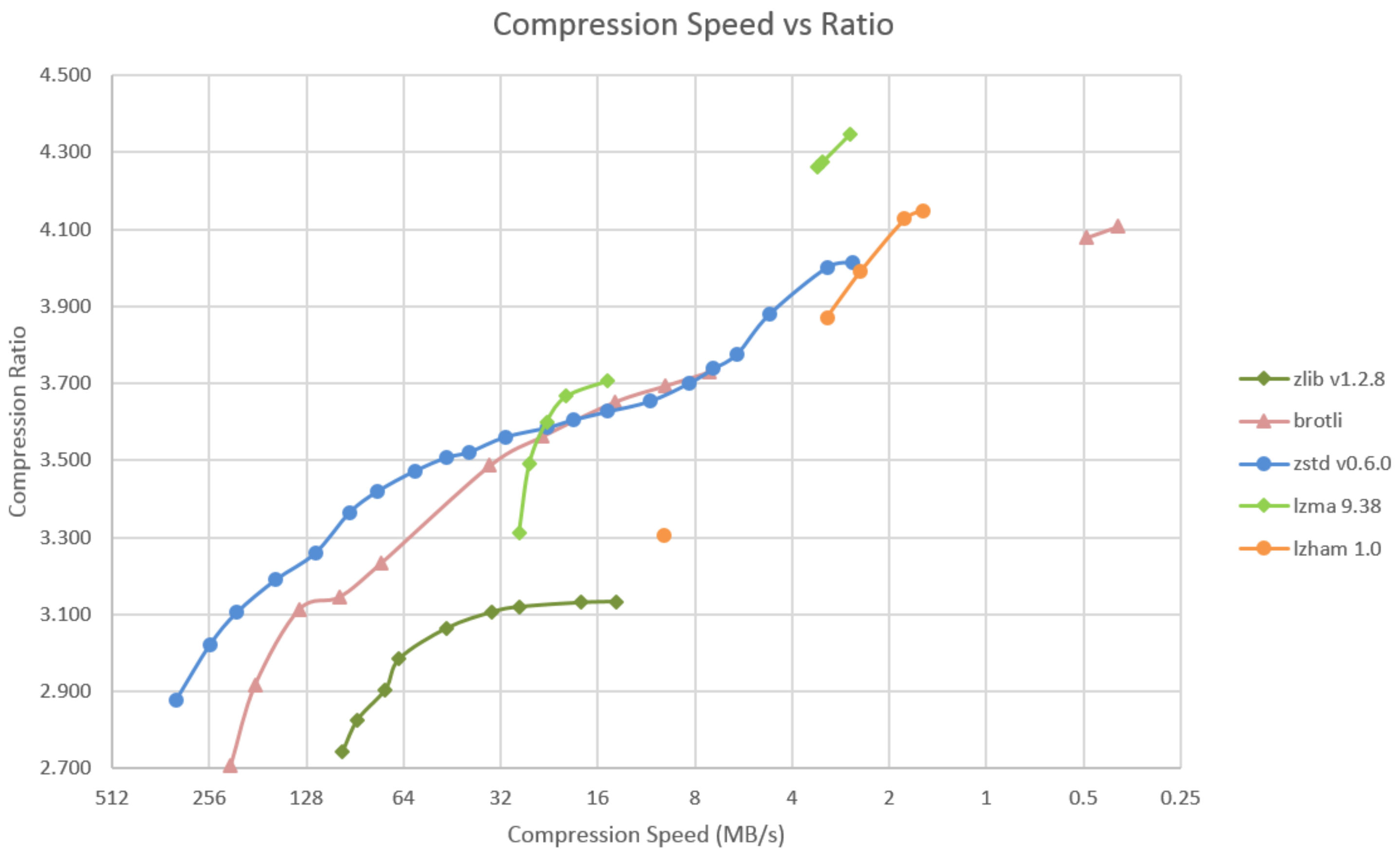
Format	Size	Encoding(s)	Decoding(s)
SAM	5 579 036 306	46	-
CRAM v2 (LZ77+Huff)	1 053 744 556	183	27
CRAM v3 (order 1 **rANS**)	869 500 447	75	31

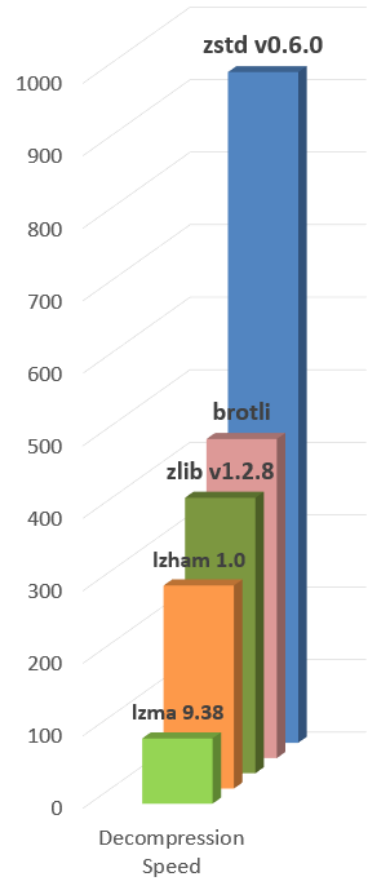
gzip used everywhere … but it's ancient … **what's next?**

Google – first **Zopfli** (gzip compatible)

Now **Brotli** (incompatible, Huffman) – Firefox support, all news:

"Google's new Brotli compressio algorithm is 26 percent better than existing solutions" (22.09.2015) … now **[ZSTD]** (tANS, Yann Collet, Facebook):





general (Apple [LZFSE], Fb [ZSTD]), DNA ([CRAM]), games ([LZNA], [BitKnit]), Google [VP10], [WebP]

Huffman coding (HC), prefix codes:
most of everyday compression,
e.g. zip, gzip, cab, jpeg, gif, png, tiff, pdf, mp3…
Zlibh (the fastest generally available implementation):
Encoding ~ **320 MB/s** (/core, 3GHz)
Decoding ~ **300-350 MB/s**

Range coding (RC): **large alphabet** arithmetic coding, needs multiplication, e.g. 7-Zip, VP Google video codecs (e.g. YouTube, Hangouts).
Encoding ~ **100-150 MB/s**
Decoding ~ **80 MB/s**

tradeoffs

(**binary**) **Arithmetic coding (AC):**
H.264, H.265 video, ultracompressors e.g. PAQ
Encoding/decoding ~ **20-30MB/s**

example of Huffman penalty
for truncated $\rho(1-\rho)^n$ distribution
(can't use less than 1 bits/symbol)

Ratio →	$8/H$	*zlibh*	*FSE*
$\rho = 0.5$	4.001	3.99	4.00
$\rho = 0.6$	4.935	4.78	4.93
$\rho = 0.7$	6.344	5.58	6.33
$\rho = 0.8$	8.851	6.38	8.84
$\rho = 0.9$	15.31	7.18	15.29
$\rho = 0.95$	26.41	7.58	26.38
$\rho = 0.99$	96.95	7.90	96.43

Huffman: 1byte → at least 1 bit
ratio ≤ 8 here

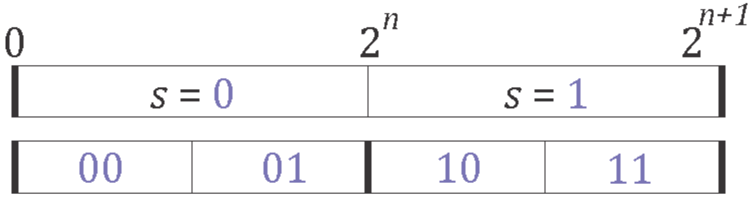
Asymmetric Numeral Systems (ANS) tabled (tANS) - without multiplication
FSE implementation of **tANS:** Encoding ~ **350 MB/s** Decoding ~ **500 MB/s**

RC → ANS: **~7x decoding speedup, no multiplication** (switched e.g. in LZA compressor)
HC → ANS means **better compression** and **~ 1.5x decoding speedup** (e.g. zhuff, lzturbo)

Operating on fractional number of bits

We have information stored in a number x and want to add information of symbol s=0,1:
asymmetrize ordinary/symmetric **binary system**: optimal for Pr(0)=Pr(1)=1/2

most significant position $x' = x + s\cdot 2^n$ | $x' = 2x + s$ least significant position



odd even

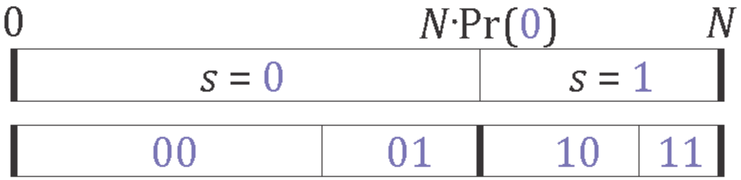
s=0 s=1

x'		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18 …
x	s=0	0		1		2		3		4		5		6		7		8		9
	s=1		0		1		2		3		4		5		6		7		8	

e.g. $x = 1 \xrightarrow{s=0} 2 \xrightarrow{1} 5 \xrightarrow{1} 11 \xrightarrow{1} 23 \xrightarrow{1} 47$

range/arithmetic coding:

rescale ranges



some **asymmetric binary system** for Pr(0) = 1/4, Pr(1) = 3/4
redefine even/odd numbers - change their densities:

$x' \approx x/\mathrm{Pr}(s)$

s=1 s=0

x'		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18 …
x	s=0	0				1				2				3				4		
	s=1		0	1	2		3	4	5		6	7	8		9	10	11		12	13

e.g. $x = 1 \xrightarrow{s=0} 4 \xrightarrow{1} 6 \xrightarrow{1} 9 \xrightarrow{1} 13 \xrightarrow{1} 18$

<table>
<tr><td>restricting range N to length L subrange
contains $\mathbf{lg}(\boldsymbol{N}/\boldsymbol{L})$ bits</td><td>number x
contains $\mathbf{lg}(\boldsymbol{x})$ bits</td></tr>
<tr><td colspan="2">adding symbol of probability p - containing $\mathbf{lg}(\mathbf{1}/\boldsymbol{p})$ bits</td></tr>
<tr><td>$\lg(N/L) + \lg(1/p) \approx \lg(L/L')$ for $\boldsymbol{L' \approx p \cdot L}$</td><td>$\lg(x) + \lg(1/p) = \lg(x')$ for $\boldsymbol{x' \approx x/p}$</td></tr>
</table>

Asymmetric numeral systems – redefine even/odd numbers:

symbol distribution: $\bar{s}: \mathbb{N} \to \mathcal{A}$ ($\mathcal{A}$ – alphabet, e.g. $\{0,1\}$)

($\bar{s}(x) = \mathrm{mod}(x, b)$ for base b numeral system: $C(s,x) = bx + s$)

Should be still **uniformly distributed** – but with **density p_s**:

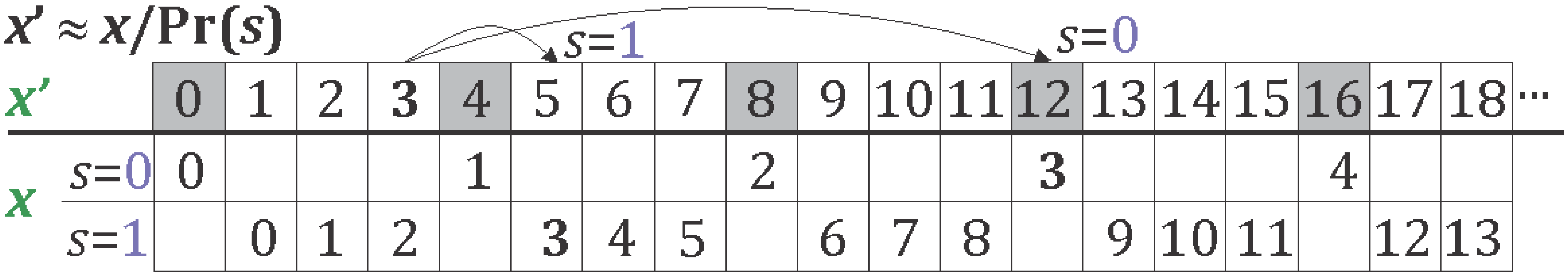
$$\#\{0 \le y < x : \bar{s}(y) = s\} \approx x p_s$$

then x becomes x-th appearance of given symbol:

$$C(s,x) = x' \;\;:\;\; \bar{s}(x') = s, \quad |\{0 \le y < x' : \bar{s}(y) = \bar{s}(x')\}| = x$$

$$D(x') = (\bar{s}(x'), \quad |\{0 \le y < x' : \bar{s}(y) = \bar{s}(x')\}|)$$

$$C(D(x')) = x' \qquad D(C(s,x)) = (s,x) \qquad \boldsymbol{x' \approx x/p_s}$$



x'		0	1	2	**3**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18…
x	s=0	0				1				2				**3**				4		
	s=1		0	1	2		**3**	4	5		6	7	8		9	10	11		12	13

e.g. $\boldsymbol{x} = 1 \xrightarrow{s=0} 4 \xrightarrow{1} 6 \xrightarrow{1} 9 \xrightarrow{1} 13 \xrightarrow{1} 18$

$\bar{s}(x) = 0$ if $\mathrm{mod}(x,4) = 0$, else $\bar{s}(x) = 1$

example: **range asymmetric binary systems (rABS)**

$\Pr(0) = 1/4 \quad \Pr(1) = 3/4$ - take base 4 system and merge 3 digits,

cyclic (0123) symbol distribution $\bar{s}$ becomes cyclic (0111):

x'	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
(s,x)	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4

→

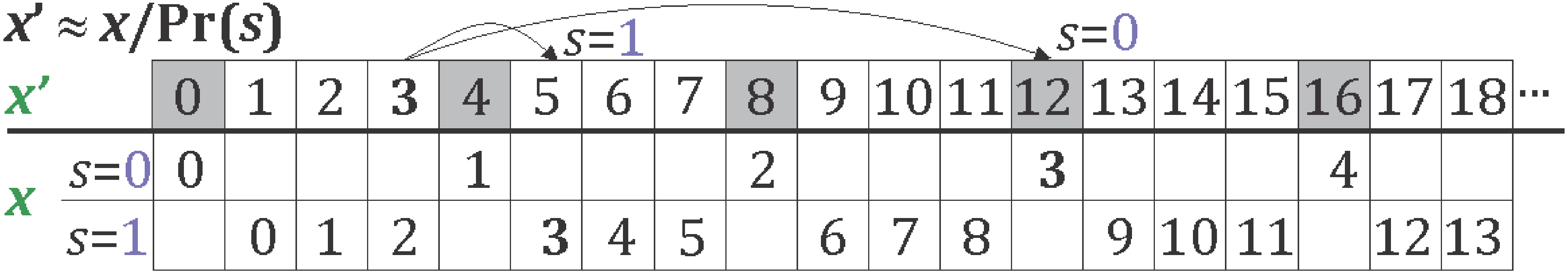
x'	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$(\bar{s},x)$	0	0	1	2	1	3	4	5	2	6	7	8	3	9	10	11	4	12	13	14

$$\bar{s}(x) = 0 \text{ if } \mathrm{mod}(x,4) = 0, \qquad \text{else } \bar{s}(x) = 1$$

to decode or encode 1, localize quadruple ($\lfloor x/4 \rfloor$ or $\lfloor x/3 \rfloor$)

$$\text{if } \bar{s}(x) = 0, \ \ D(x) = (0, \lfloor x/4 \rfloor) \ \ \text{else} \ \ D(x) = (1, 3\lfloor x/4 \rfloor + \mathrm{mod}(x,4) - 1)$$

$$C(0,x) = 4x \qquad\qquad C(1,x) = 4\lfloor x/3 \rfloor + 1 + \mathrm{mod}(x,3)$$



x'		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	⋯
x	s=0	0				1				2				3				4			
	s=1		0	1	2		3	4	5		6	7	8		9	10	11		12	13	

e.g. $\quad x = 1 \xrightarrow{s=0} 4 \xrightarrow{1} 6 \xrightarrow{1} 9 \xrightarrow{1} 13 \xrightarrow{1} 18$

rANS - range variant for large alphabet $\mathcal{A} = \{0, .., m-1\}$

assume $\Pr(s) = f_s / 2^n$ $c_s := f_0 + f_1 + \cdots + f_{s-1}$

start with base 2^n numeral system and merge length f_s ranges

for $x \in \{0,1,\dots,2^n-1\}$, $\bar{\boldsymbol{s}}(\boldsymbol{x}) = \mathbf{max}\{\boldsymbol{s} : \boldsymbol{c_s} \le \boldsymbol{x}\}$, $mask = 2^n - 1$

encoding: $C(s,x) = \lfloor x/f_s \rfloor \ll n + \mathrm{mod}(x, f_s) + c_s$

decoding: $s = \bar{s}(x \,\&\, mask)$ (e.g. tabled, alias method)

$$D(x) = (s,\ f_s \cdot (x \gg n) + (x \,\&\, mask) - c_s)$$

Plus renormalization **to make for example** $x \in \{2^{16}, \dots, 2^{32}-1\},\ n = 12$:

Decoding step ($mask = 2^n - 1$)	**Encoding step** (msk = 2^{16} - 1)
s = symbol[x & $mask$]	s = readSymbol();
writeSymbol(s);	if(x > bound[s])
x = f[s] (x >> n) + (x & $mask$) – c[s];	{write16bits(x & msk); x >>= 16; }
if($x < 2^{16}$) $x = x$ << 16 + read16bits();	x = (x / f[s]) << n + (x % f[s]) + c[s];

CRAM v3 (2015): order 1 rANS: 256 frequencies depending on previous symbol

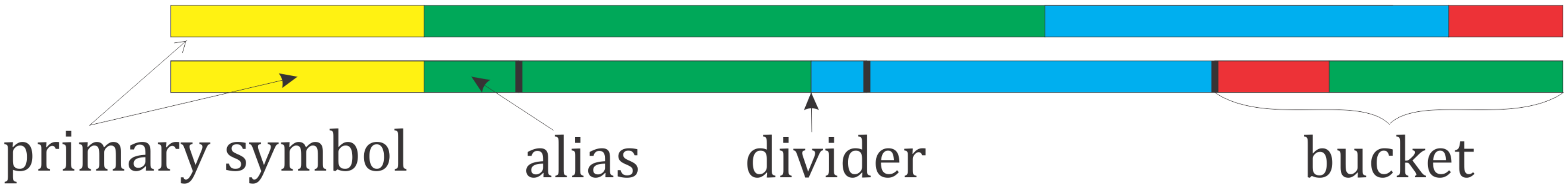
Plus renormalization to make for example $x \in \{2^{16}, \ldots, 2^{32}-1\}$, $n = 12$:

Decoding step $(mask = 2^n - 1)$	**Encoding step** $(msk = 2^{16} - 1)$
s = symbol(x & $mask$);	s = readSymbol();
writeSymbol(s);	if(x > bound[s])
x = f[s] (x >> n) + (x & $mask$) – **CDF**[s];	{write16bits(x & msk); x >>= 16; }
if($x < 2^{16}$) $x = x$ << 16 + read16bits();	x = (x / f[s]) << n + (x % f[s]) + **CDF**[s];

Similar to **Range Coding**, but decoding has 1 multiplication (instead of 2), for determining symbol range is fixed, and state is 1 number (instead of 2), making it convenient for SIMD vectorization (https://github.com/rygorous/ryg_rans).

Various ways to handle symbol$(y) = s$: $CDF[s] \le y < CDF[s+1]$ for $0 \le y < 2^n$:
Tabled(symbol[y]), **search** (binary or SIMD - CDF only!), or **alias method**:

'Alias' method: rearrange probability distribution into m buckets:
containing the primary symbol and eventually a single 'alias' symbol



uABS - uniform binary variant ($\mathcal{A}=\{0,1\}$) - extremely accurate

Assume binary alphabet, $\boldsymbol{p} \coloneqq \mathbf{Pr}(\mathbf{1})$, denote $\boldsymbol{x_s} = \{\boldsymbol{y} < \boldsymbol{x}: \overline{\boldsymbol{s}}(\boldsymbol{y}) = \boldsymbol{s}\} \approx \boldsymbol{x}\boldsymbol{p_s}$

For uniform symbol distribution we can choose:

$$\boldsymbol{x_1} = \lceil \boldsymbol{xp} \rceil \qquad \boldsymbol{x_0} = \boldsymbol{x} - \boldsymbol{x_1} = \boldsymbol{x} - \lceil \boldsymbol{xp} \rceil$$

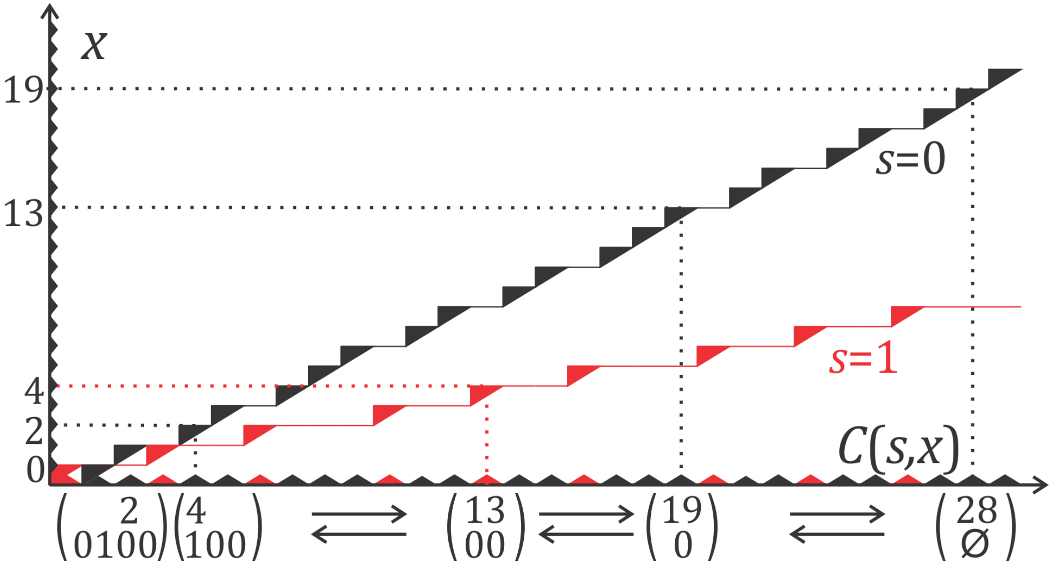
$\overline{s}(x) = 1$ if there is jump on next position: $s = \overline{s}(x) = \lceil (x+1)p \rceil - \lceil xp \rceil$

decoding function: $D(x) = (s, x_s)$

its inverse – **coding function:**

$$C(0,x) = \left\lceil \frac{x+1}{1-p} \right\rceil - 1$$

$$C(1,x) = \left\lfloor \frac{x}{p} \right\rfloor$$



For $p = \Pr(1) = 1 - \Pr(0) = 0.3$:

$\boldsymbol{C(s,x) \approx x/\Pr(s)}$

s=0 s=1

$C(s,x)$	0	1	2	3	4	**5**	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	···
x s=0		0	1		2	3		4	**5**	6		7	8		9	10		11	12	13		14	15		
s=1	0			1			2				3			4			**5**				6			7	
px	0.0			0.9			1.8				3.0			3.9			4.8				6.0			6.9	

Stream version – renormalization

Up to now: we encode using succeeding C functions into a huge number x,
then decode (in opposite direction!) using succeeding D.
Like in arithmetic coding, we need **renormalization** to limit working precision -
enforce $\boldsymbol{x \in I = \{L, \ldots, bL-1\}}$ by transferring base-$\boldsymbol{b}$ youngest digits:

ANS **decoding** step from state x	**encoding** step for symbol s from state x
$(s, x) = D(x)$; useSymbol(s); **while $\boldsymbol{x < L}$, $\boldsymbol{x = bx + \mathrm{readDigit}()}$;**	**while $\boldsymbol{x \geq maxX[s]}$** // $= bL_s$ **{writeDigit(mod($\boldsymbol{x, b}$)); $\boldsymbol{x = \lfloor x/b \rfloor}$};** $x = C(s, x)$;

For unique decoding,
we need to ensure that there is a single way to perform above loops:

$$I = \{L, \ldots, bL-1\},\ I_s = \{L_s, \ldots, bL_s - 1\} \quad \text{where} \quad I_s = \{x : C(s,x) \in I\}$$

Fulfilled e.g. for
- rABS/rANS when $p_s = f_s/2^n$ has $1/L$ accuracy: 2^n divides L,
- uABS when p has $1/L$ accuracy: $b\lceil Lp\rceil = \lceil bLp\rceil$,
- in tabled variants (tABS/tANS) it will be fulfilled by construction.

uABS

```
xp = (uint64_t)x * p0;
out[i]=((xp & mask) >= p0);
xp >>= 16;
x = out[i] ? xp : x - xp;
```

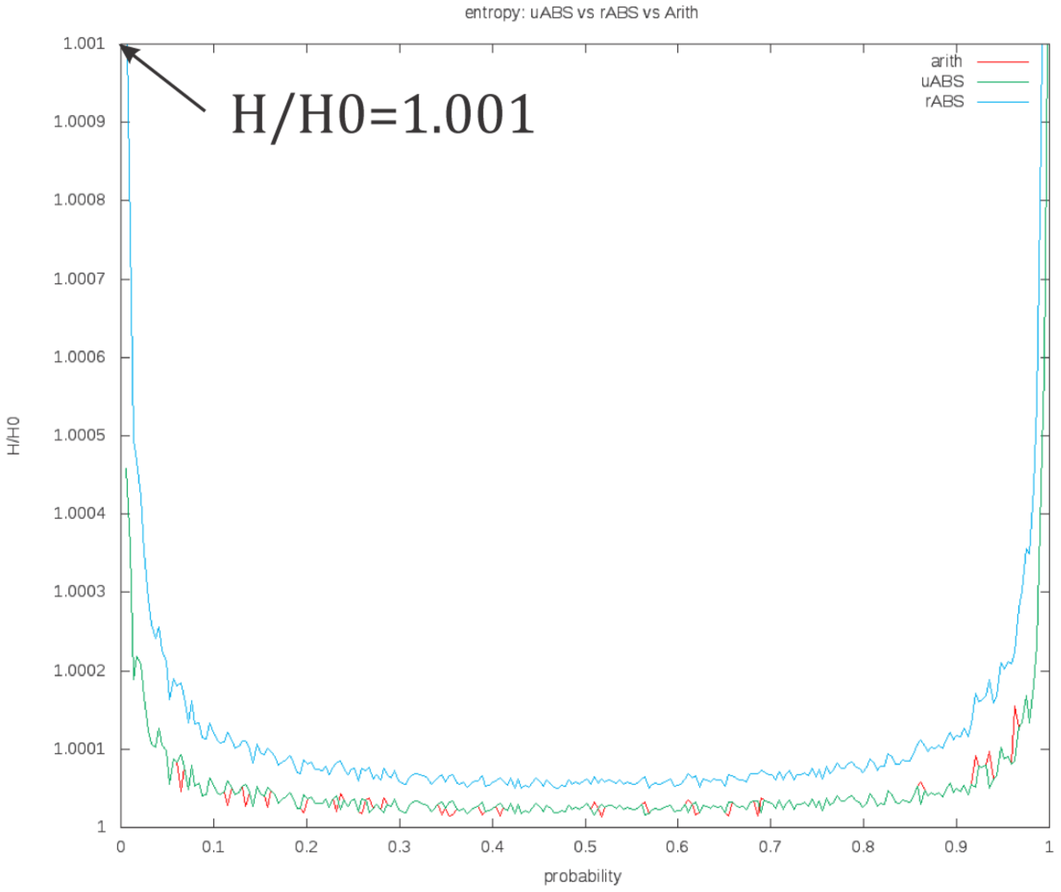
rABS

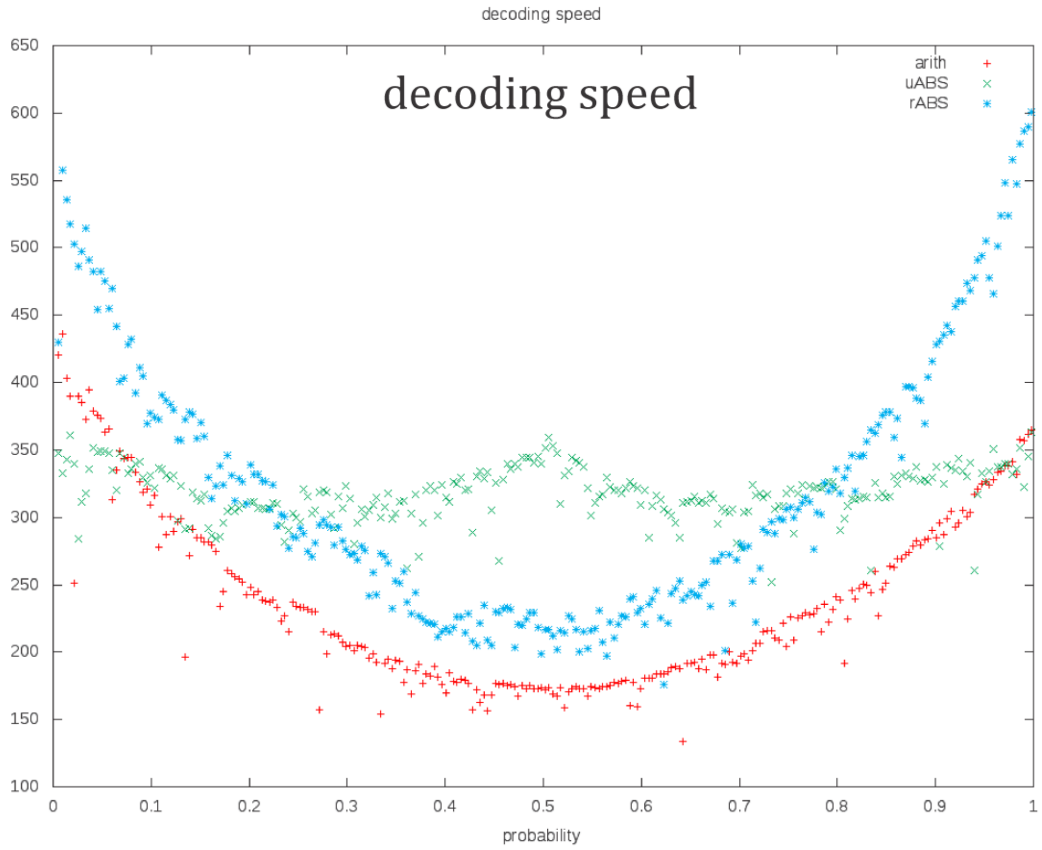
```
xf = x & mask;         //32bit
xn = p0 * (x >> 16);
if (xf < p0)
{ x = xn + xf; out[i] = 0 }
else {x-=xn+p0;out[i] = 1 }
```

```
     if (x < 0x10000) { x = (x << 16) | *ptr++;  }

//32bit x, 16bit renormalization, mask = 0xffff
```

arithmetic coding

```
split = low + ((uint64_t)
  (hi - low) * p0 >> 16);
out[i] = (x > split);
if (out[i]) {low = split + 1}
   else {hi = split;}

if ((low ^ hi) < 0x10000)
{  x = (x << 16) | *ptr++;
   low <<= 16;
   hi = (hi << 16) | mask }
```



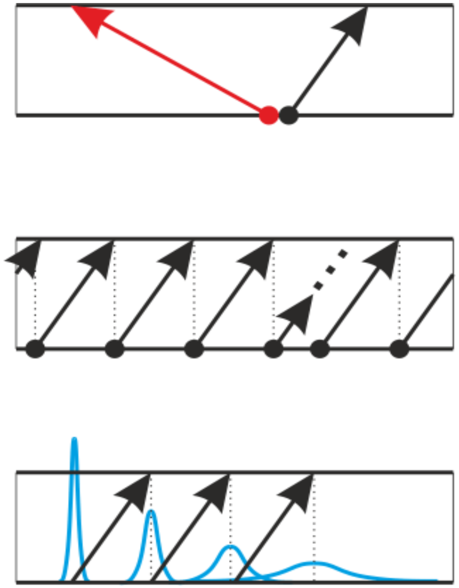


RENORMALIZATION to prevent $x \to \infty$
Enforce $x \in I = \{L, \dots, 2L-1\}$ by
transmitting lowest bits to bitstream

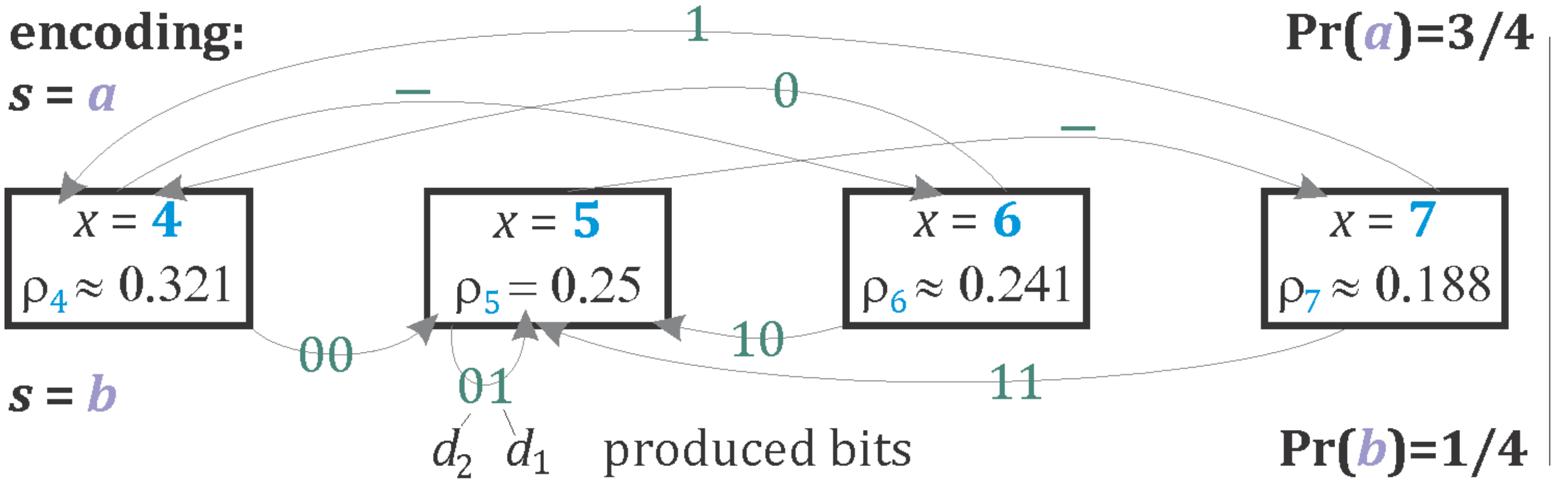
"buffer" x contains $\lg(x)$ bits of information
– produces bits when **accumulated**

Symbol of $Pr = p$ contains $\lg(1/p)$ bits:

$\lg(x) \to \lg(x) + \lg(1/p)$ modulo 1



Tabled variant (tABS/tANS) – put everything into a table:



encoding:

s = a

$x = 4$, $\rho_4 \approx 0.321$

$x = 5$, $\rho_5 = 0.25$

$x = 6$, $\rho_6 \approx 0.241$

$x = 7$, $\rho_7 \approx 0.188$

1 – 0 –

s = b

00 01 10 11

d_2 d_1 produced bits

Pr(a)=3/4

Pr(b)=1/4

decoding:

$x \to s$, new x

4 $\to$ a, 6+d_1
5 $\to$ b, 4 + 2d_2+ d_1
6 $\to$ a, 4
7 $\to$ a, 5

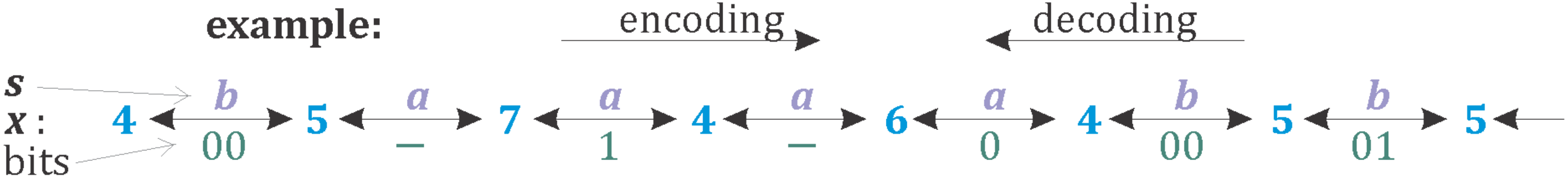
newX *nbBits*
decodingTable

---

example:

encoding →

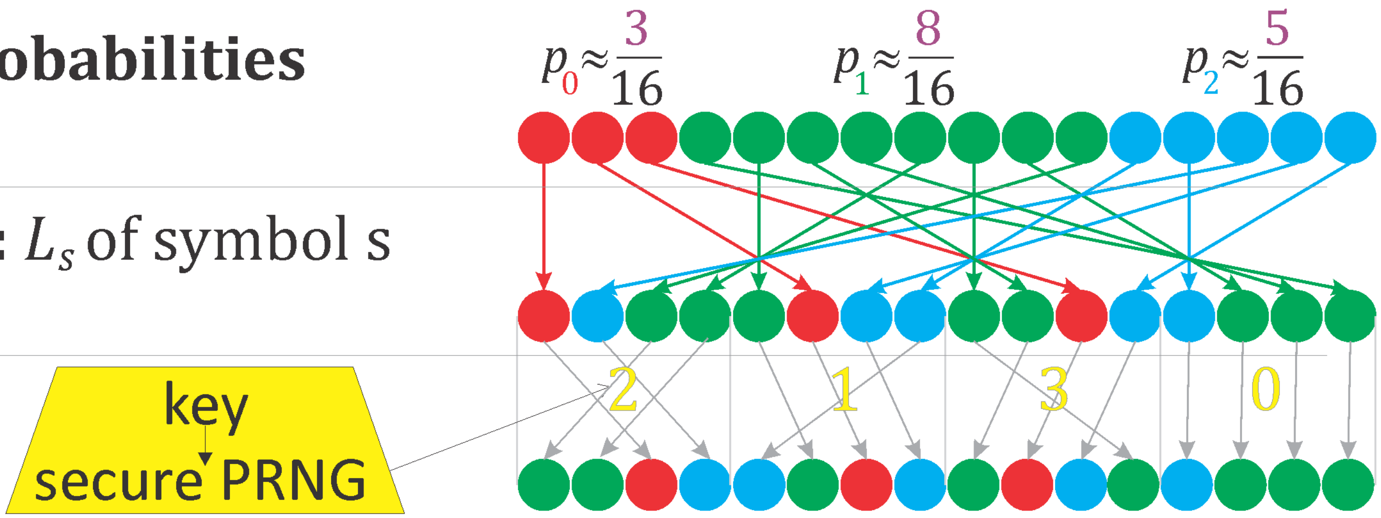
← decoding



s		b		a		a		a		a		b		b	
x :	4		5		7		4		6		4		5		5
bits		00		–		1		–		0		00		01	

example of tANS construction for L=16 states and size 3 alphabet

t=decodingTable[x]; use(t.symbol); $\mathbf{x} \rightarrow$ **t.newX + readBits(t.nbBits);**



1. Approximate probabilities

as $p_s \approx L_s / L$

2. Spread symbols: L_s of symbol s

(fast, step = 5)

2*. Scramble

(4 block cycle)

3. Enumerate appearances

from L_s to $2L_s$ - 1

L =16, L_0= 3, L_1= 8, L_2= 5

$C(s,x)$		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
x	s=0			3				4			5						
	s=1	8	9				10			11			12		13	14	15
	s=2				5	6			7			8		9			

4. Renormalize to make x remain in I = {L, ..., 2L-1} range

decodingTable:

(**symbol**,

nbBits,

newX)

x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
symbol	1	1	0	2	2	1	0	2	1	0	2	1	2	1	1	1
pre-renormalization x_{tmp}	8	9	3	5	6	10	4	7	11	5	8	12	9	13	14	15
nbBits to read to return to I	1	1	3	2	2	1	2	2	1	2	1	1	1	1	1	1
newX = x_{tmp} << nbBits	16	18	24	20	24	20	16	28	22	20	16	24	18	26	28	30

5. Endode/decode - e.g. decoding |11|10|0|0|011|0|1|01|0|0|1|1|

bits

$$x:\ 25 \xrightarrow[0]{11} 23 \xrightarrow[2]{10} 30 \xrightarrow[1]{0} 28 \xrightarrow[2]{0} 18 \xrightarrow[0]{011} 27 \xrightarrow[1]{0} 24 \xrightarrow[1]{1} 23 \xrightarrow[2]{01} 29 \xrightarrow[1]{0} 26 \xrightarrow[2]{0} 16 \xrightarrow[1]{1} 17 \xrightarrow[1]{1} 19$$

s

Method 1 tANS decoding step, $X = x - L \in \{0, \ldots, L-1\}$

```
$t = decodingTable[X]$            $\{ X \in \{0,..,L-1\}$ is current state $\}$
useSymbol($t.symbol$)             $\{$ use or store decoded symbol $\}$
$X = t.newX +$readBits($t.nbBits$)          $\{$ state transition $\}$
```

Method 2 tANS encoding step for symbol s and state $x = X + L$

```
$nbBits = (x + nb[s]) >> r$          $\{2^r = 2L\}$
$useBits(x, nbBits)$          $\{$use $nbBits$ of the youngest bits of $x\}$
$x = encodingTable[start[s] + (x >> nbBits)]$
```

Method 3 Preparation for tANS decoding, $L = 2^R$

```
Require: $next[s] = L_s$      $\{$number of next appearance of symbol $s\}$
  for $X = 0$ to $L - 1$ do
    $t.symbol = symbol[X]$       $\{$ symbol is from spread function $\}$
    $x = next[t.symbol] + +$        $\{ D(X + L) = (symbol, x) \}$
    $t.nbBits = R - \lfloor \lg(x) \rfloor$       $\{$ number of bits$\}$
    $t.newX = (x << t.nbBits) - L$        $\{$ properly shift $x$ $\}$
    $decodingTable[X] = t$
  end for
```

Method 4 Preparation for tANS encoding, $L = 2^R$, $r = R + 1$

```
Require: $k[s] = R - \lfloor \lg(L_s) \rfloor$              $\{nbBits = k[s]$ or $k[s] - 1\}$
Require: $nb[s] = (k[s] << r) - (L_s << k[s])$
Require: $start[s] = -L_s + \sum_{s'<s} L_{s'}$
Require: $next[s] = L_s$
  for $x = L$ to $2L - 1$ do
    $s = symbol[x - L]$
    $encodingTable[start[s] + (next[s] + +)] = x;$
  end for
```

Method 7 An example of fast symbol spread function [11]

```
$X = 0; step = 5/8L + 3$     $\{$ some initial position and choice of step$\}$
for $s = 0$ to $m - 1$ do
    for $i = 1$ to $L_s$ do
        $symbol[X] = s; X =$mod$(X + step, L)$
    end for
end for
```

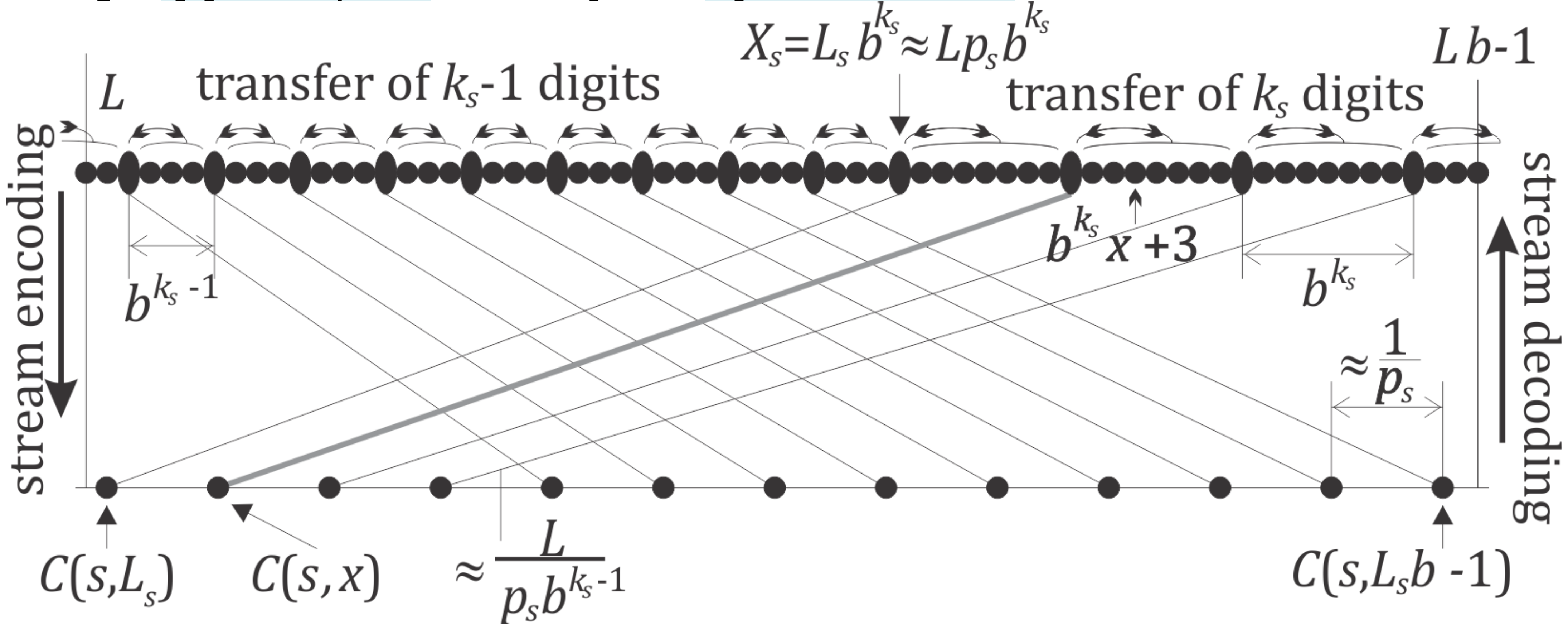
Single step of stream version:

to get $x \in I$ to $I_s = \{L_s, \dots, bL_s - 1\}$, we need to transfer k digits:

$$x \overset{s}{\to} \left(C\left(s, \lfloor x/b^k \rfloor\right),\ \operatorname{mod}(x, b^k)\right) \qquad \text{where} \qquad k = \lfloor \log_b(x/L_s) \rfloor$$

$$k = k_s \quad \text{or} \quad k = k_s - 1 \qquad \text{for} \qquad \boldsymbol{k_s = -\lfloor \log_b(p_s) \rfloor} = -\lfloor \log_b(L_s/L) \rfloor$$

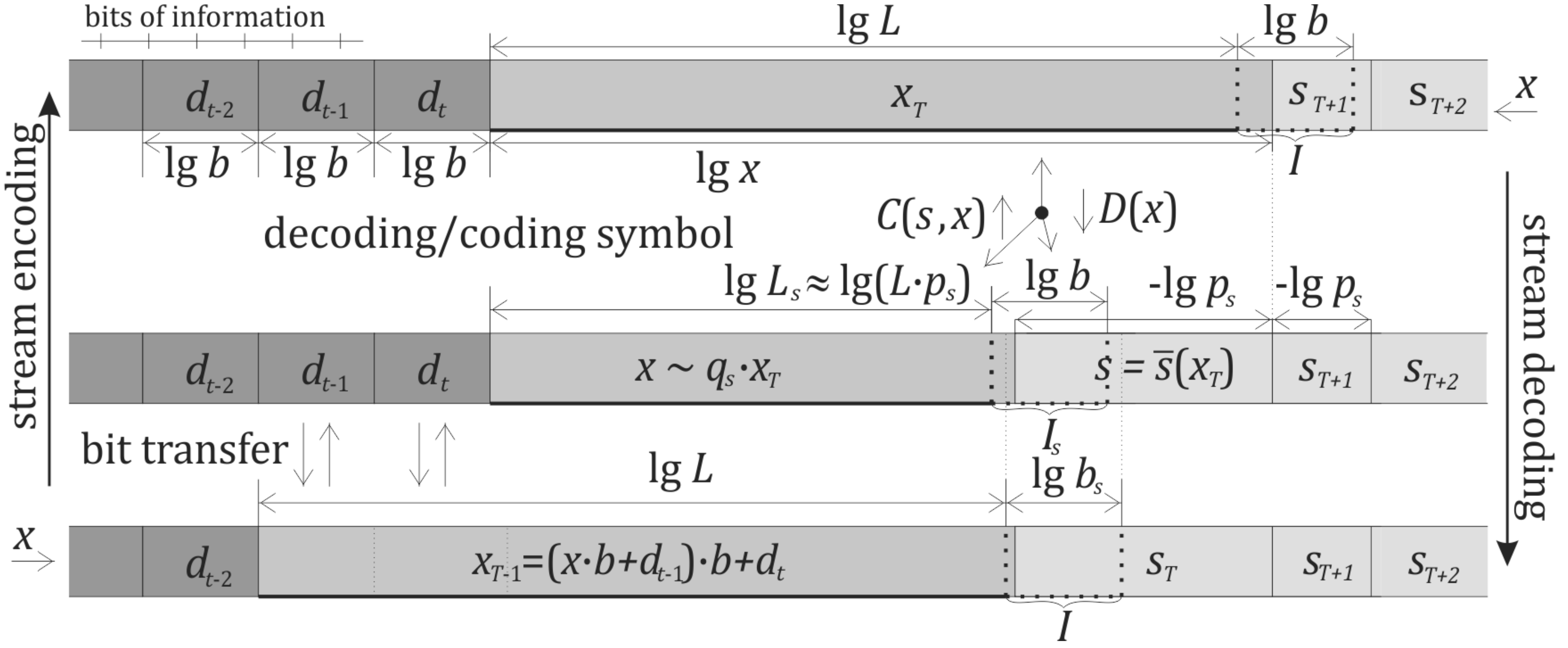
e.g.: $\boldsymbol{p_s = 13/66}, \boldsymbol{b = 2}, k_s = 3, L_s = 13, L = 66,\ b^{k_s}x + 3 = 115, x = 14$:



Huffman: $k = -\lg(p_s) = k_s$, above lines are vertical

General picture:

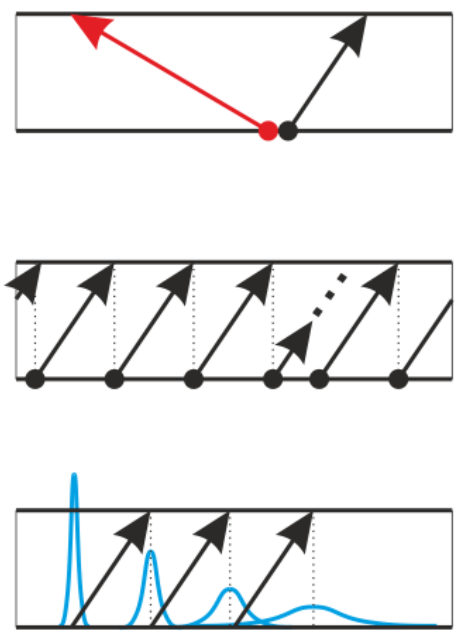
encoder prepares before consuming succeeding symbol
decoder produces symbol, then consumes succeeding digits



Decoding is in **opposite direction**: we have **stack of symbols** (LIFO)

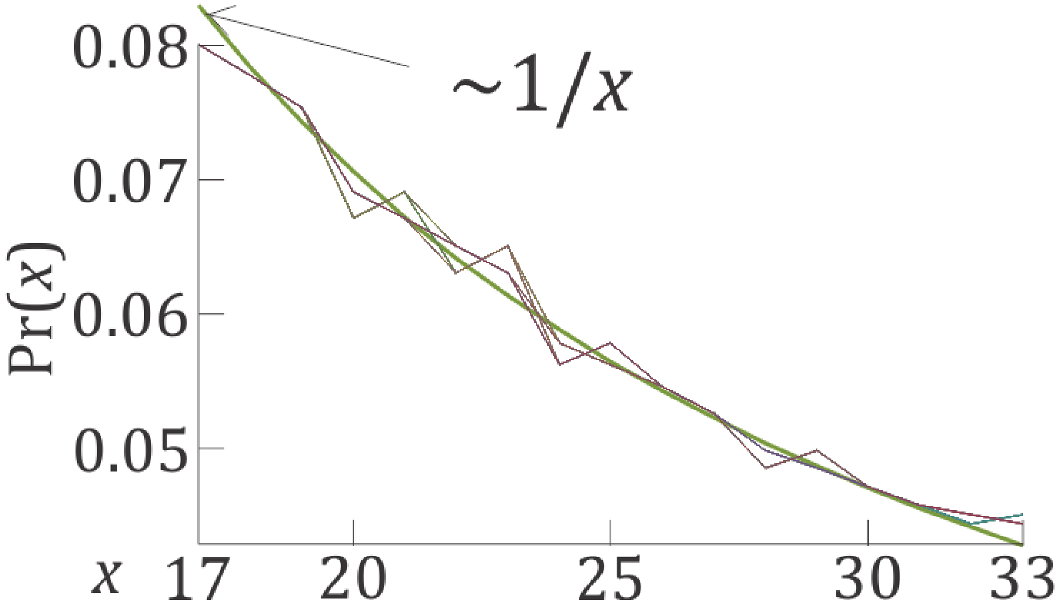
- encoding should be made in backward direction (buffer required),
- the final state has to be stored, but we can write information in initial state, alternatively: fixing this state will make it a checksum – random after an error.

In single step ($I = \{L, \dots, bL-1\}$): $\mathbf{lg}(\boldsymbol{x}) \to \approx \mathbf{lg}(\boldsymbol{x}) + \mathbf{lg}(\mathbf{1}/\boldsymbol{p})$ **modulo** $\mathbf{lg}(\boldsymbol{b})$



Three sources of **unpredictability/chaosity**:

1) **Asymmetry**: behavior strongly dependent on chosen symbol – small difference changes decoded symbol and so the entire behavior.
2) **Ergodicity**: usually $\log_b(1/p)$ is irrational – succeeding iterations cover entire range.
3) **Diffusivity**: $C(s,x)$ is close but not exactly x/p_s – there is additional ‘diffusion’ around expected value



So $\mathbf{lg}(\boldsymbol{x}) \in [\lg(L), \lg(L) + \lg(b))$ **has nearly uniform distribution** – x has approximately:

$$\mathbf{Pr}(\boldsymbol{x}) \propto \mathbf{1}/\boldsymbol{x}$$

probability distribution – contains $\lg(1/\Pr(x)) \approx \lg(x) + \text{const}$ bits of information.

What **symbol spread** should we choose? (link)
(using PRNG seeded with cryptkey for encryption)

for example: **rate loss** for $p = (0.04, \quad 0.16, \quad 0.16, \quad 0.64)$
$L = 16$ states, $q = (1, 3, 2, 10)$, $\quad q/L = (0.0625, 0.1875, 0.125, 0.625)$

method	symbol spread	dH/H rate loss	
-	-	**~0.011**	penalty of **quantizer** itself
Huffman	**0011222233333333**	**~0.080**	would give **Huffman** decoder
spread_range_i()	**0111223333333333**	**~0.059**	**Increasing** order
spread_range_d()	**3333333333221110**	**~0.022**	**Decreasing** order
spread_fast()	**0233233133133133**	**~0.020**	**fast**
spread_prec()	**3313233103332133**	**~0.015**	**close to quantization** dH/H
spread_**tuned**()	**3233321333313310**	**~0.0046**	**better than quantization** dH/H due to using also p
spread_**tuned**_s()	**2333312333313310**	**~0.0040**	L log L complexity (sort)
spread_**tuned**_p()	**2331233330133331**	**~0.0058**	testing 1/(p ln(1+1/i)) ~ i/p approximation

Precise initialization (heuresis)

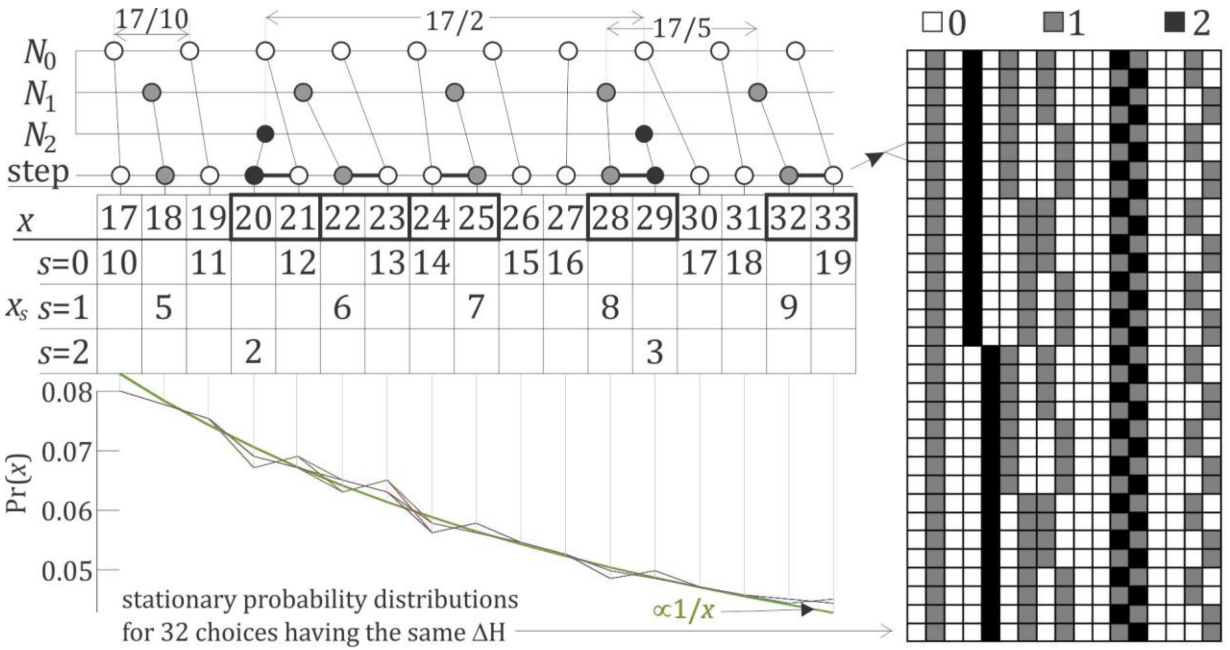
$$N_s = \left\{\frac{0.5+i}{p_s} : i = 0, \ldots, L_s - 1\right\}$$

are uniform – we need to shift them to natural numbers.
(priority queue with put, getmin)

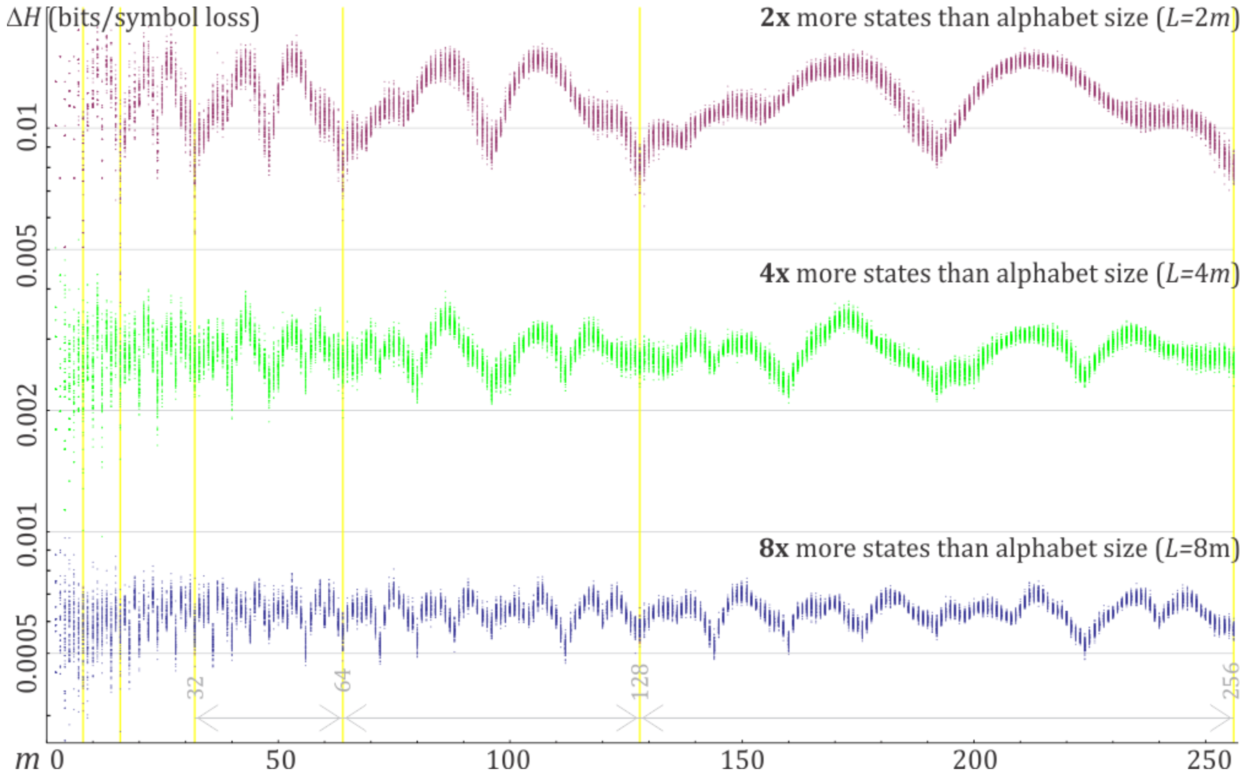
```
for $s = 0$ to $n - 1$ do
    put($(0.5/p_s, s)$);
for $X = 0$ to $L - 1$ do
    {$(v, s)$ = getmin;
    put($(v + 1/p_s, s)$);
    $symbol[X] = s$;
    }
```

approximately:

$$\boldsymbol{\Delta H} \ \propto \ \left(\frac{\textbf{alphabet size}}{\boldsymbol{L}}\right)^{\mathbf{2}}$$



x	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
x_s s=0	10		11		12		13	14		15	16			17	18		19
s=1		5				6			7			8				9	
s=2				2									3				



Tuning: $p_s \approx q_s/L$. Can we "tune" spread of q_s symbols accordingly to p_s?

Shift symbols right when $p_s < q_s/L$, left otherwise

Assume $\mathbf{Pr}(x) \approx \dfrac{1}{x\ln(2)}$

$$\sum_{x=a\ldots b} \Pr(x) \approx \lg\left(\frac{b}{a}\right)$$

s appears q_s times: $i \in I_s = \{q_s, \ldots, 2q_s - 1\}$

$$\Pr(i\text{-th interval}) \approx \lg\left(\frac{i+1}{i}\right)$$

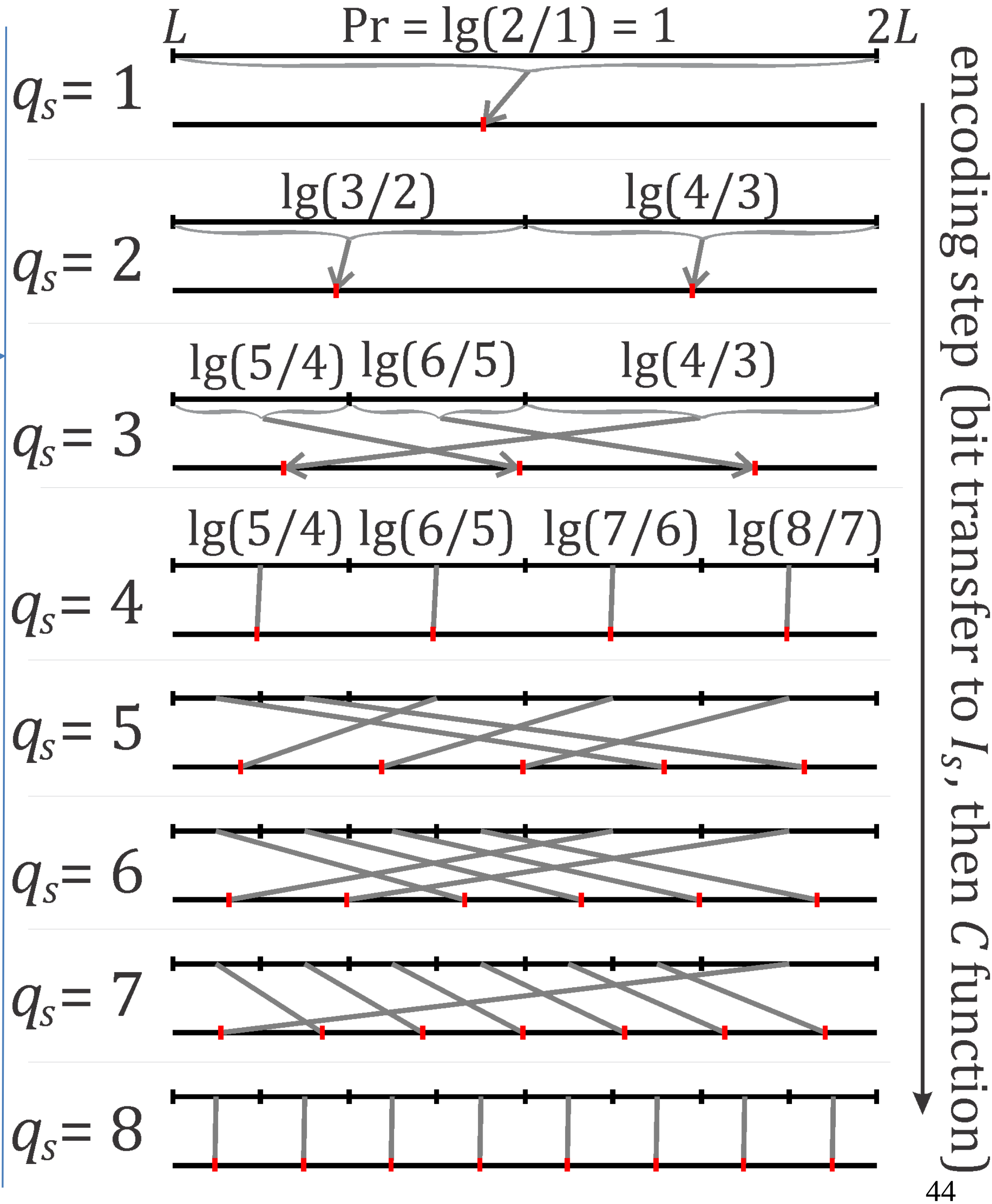
To fulfill the $\Pr(x)$ assumption,
x for this interval should fulfill:

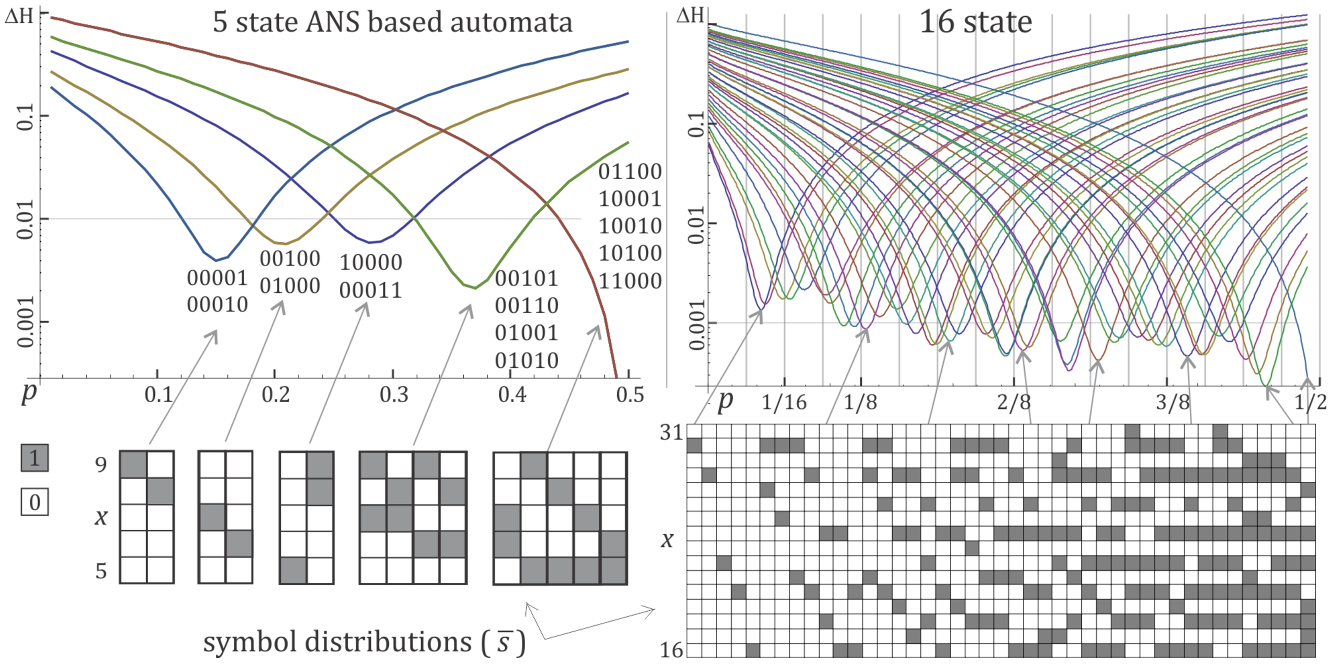
$$\lg\left(\frac{i+1}{i}\right) \cdot p_s \approx \frac{1}{x\ln(2)}$$

$$x \approx \frac{1}{p_s \ln(1 + 1/i)}$$

is the preferred position for
$i \in I_s = \{q_s, \ldots, 2q_s - 1\}$ appearance
of $p_s \approx q_s/L$ symbol

https://github.com/JarekDuda/AsymmetricNumeralSystemsToolkit





tABS

test all possible
symbol distributions
for binary alphabet

store tables for
quantized probabilities (p)
e.g. $16 \cdot 16 = 256$ bytes
← for 16 state, $\Delta H \approx 0.001$

H.264 “M decoder” (arith. coding)

```
// interval subdivision
    R_LPS = RTAB[m][(R >> 6) & 3]
    R_MPS = R - R_LPS
    if (V < R_MPS)
        R = R_MPS,   value = valMPS
    else
        V = V - R_MPS,   value = !valMPS
        R = R_LPS

// renormalization
    while (R < 2^8)
        R = R << 1
        V = V << 1
        V = V | read_one_bit()
```

tABS

$t = decodingTable[p][X];$

$X = t.newX + \text{readBits}(t.nbBits);$

$\text{useSymbol}(t.symbol);$

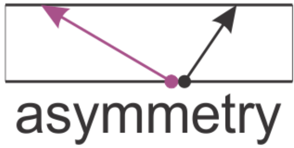
no branches,
no bit-by-bit renormalization
state is single number (e.g. SIMD)

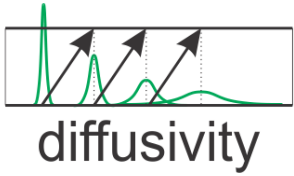
Additional tANS advantage – simultaneous encryption

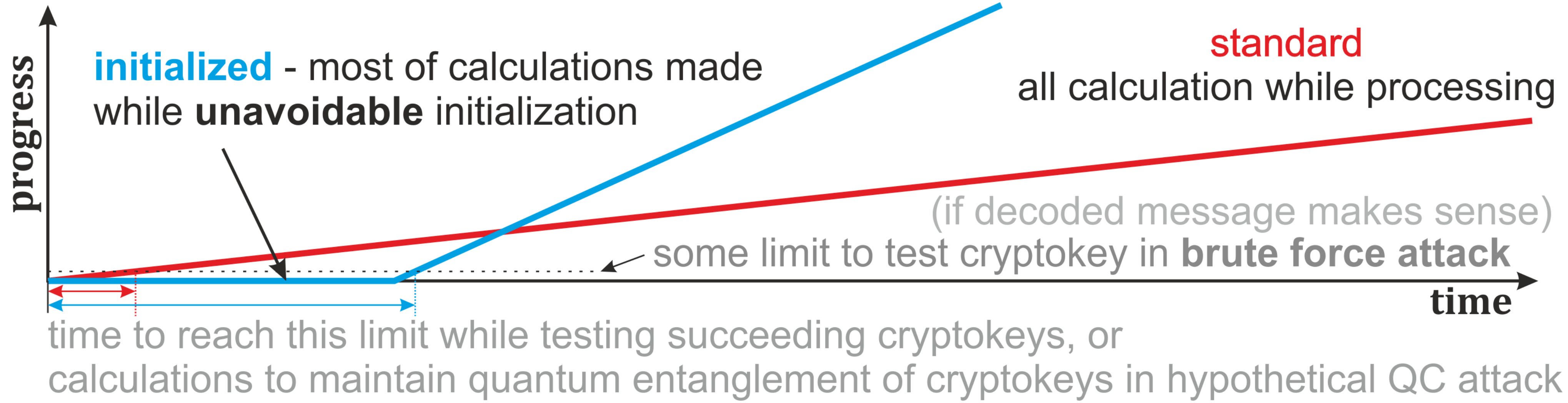
we can use huge freedom while **initialization**: choosing symbol distribution –
slightly disturb $\overline{s}(x)$ using PRNG initialized with cryptographic key

ADVANTAGES comparing to standard (symmetric) cryptography:

	standard, e.g. DES, AES	ANS based cryptography (initialized)
based on	XOR, permutations	highly **nonlinear** operations
bit blocks	fixed length	pseudorandomly **varying lengths**
"brute force" or QC attacks	just start decoding to test cryptokey	**perform initialization first** for new cryptokey, fixed to need e.g. 0.1s
speed	online calculation	most calculations while initialization
entropy	operates on bits	operates on **any input distribution**







initialized - most of calculations made while **unavoidable** initialization

standard
all calculation while processing

(if decoded message makes sense)
some limit to test cryptokey in **brute force attack**

time to reach this limit while testing succeeding cryptokeys, or
calculations to maintain quantum entanglement of cryptokeys in hypothetical QC attack

tANS (2007) - fully tabled: Apple LZFSE, Facebook ZSTD, lzturbo
fast: no multiplication (**FPGA**!), less memory efficient (~8kB for 2048 states)
static in ~32kB blocks, costly to update (rather needs rebuilding),
allows for simultaneous encryption (PRNG to perturb symbol spread)

tANS decoding step	**Encoding step** (for symbol s)
t = decodingTable[x];	nbBits = (x + nb[s]) >> r ;
writeSymbol(t.symbol);	writeBits(x, nbBits);
x = t.newX + readBits(t.nbBits);	x = encodingTable[start[s] + (x >> nbBits)];

rANS (2013) – needs multiplication – CRAM (DNA), VP10 (video), LZNA, BitKnit
more memory effective – especially for large alphabet and precision (CDF only)
better for adaptation ($CDF[s] \le y < CDF[s+1]$ - tabled, alias, binary search/SIMD)

rANS decoding step $(mask = 2^n - 1)$	**Encoding step** (s) ($msk = 2^{16} - 1$)
s = **symbol**(x & $mask$); writeSymbol(s);	if(x > bound[s])
x = f[s] (x >> n) + (x & $mask$) – **CDF**[s];	{write16bits(x & msk); x >>= 16; }
if($x < 2^{16}$) x = x << 16 + read16bits();	x = (x / f[s]) << n + (x % f[s]) + **CDF**[s];

i5-4300U: FSE/tANS: 295/467, rANS: 221/342, zlibh: 225/210 MB/s

General scheme **for Data Compression … data encoding:**

1) Transformations, predictions

To decorrelate data, make it more predictable, encode only new information

Delta coding, YCrCb, Fourier, Lempel-Ziv, Burrows-Wheeler, MTF, RLC …

2) If lossy, quantize coefficients (e.g. $c \to \text{round}(c/Q)$)

sequence of ↓ **(context_ID, symbol)**

3) Statistical modeling of final events/symbols

Static, parametrized, stored in headers, context-dependent, adaptive

5) (Entropy) coding: use $\lg(1/p_s)$ bits, $H = \sum_s p_s \lg(1/p_s)$ bits total

Prefix/Huffman: fast/cheap, but inaccurate ($p \sim 2^{-r}$)

Range/arithmetic: costly (multiplication), but accurate

Asymmetric Numeral Systems – fast and accurate

general (Apple LZFSE, Fb ZSTD), DNA (CRAM), games (LZNA, BitKnit), Google VP10, WebP