ANS coding replacing Huffman and AC – from introduction to patent issues

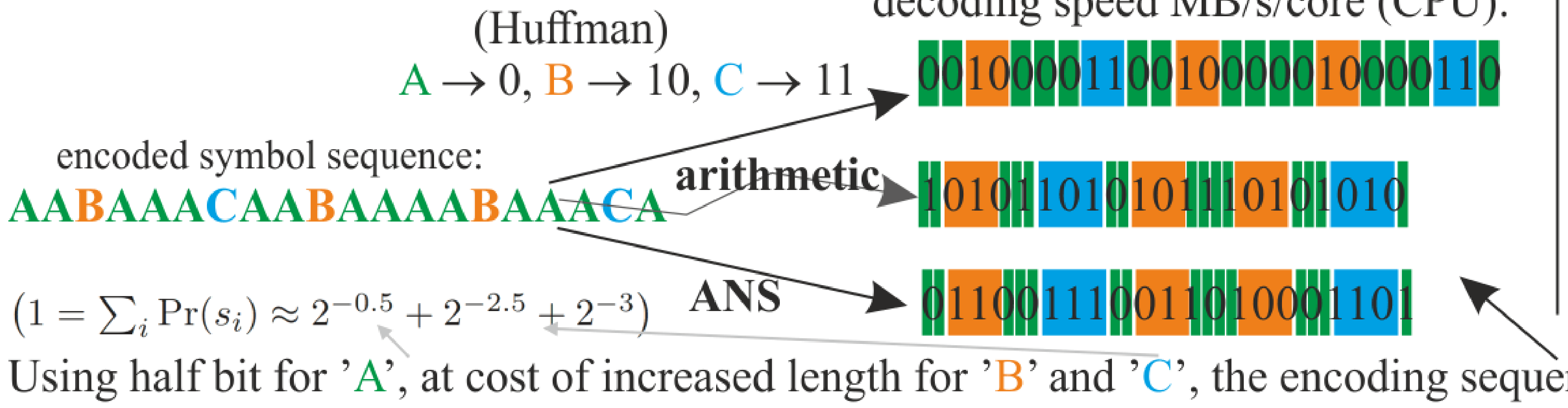
Asymmetric Numeral Systems: **symbols → final bits** of modern data compressors e.g.:

Apple LZFSE (**default in iPhones and Macs**), Facebook **ZSTD**: (e.g. in **Linux kernel**), CRAM 3.0 (**default DNA**), Google 3D Draco (e.g. Pixar), neural-network-based **JPEG XL** **~3x smaller than JPEG**, alpha, HDR, lossless – recently ISO standard and maaaany others since 2007 – **large community just sharing work and ideas**

… however, it also brought many **patent vultures** - pursuing **monopoly** …
Arithmetic coding (AC) paralyzed by dozens of patents for decades (1976 … ~2004)
How to defend from somebody patenting your work e.g. from GitHub, article???
What if a corporation is granted monopoly for method widely used for years???

Symbol carries **lg$_2$(1/Pr(symbol)) bits** (can be fractional):

Jarek Duda



Using half bit for 'A', at cost of increased length for 'B' and 'C', the encoding sequence is shorter

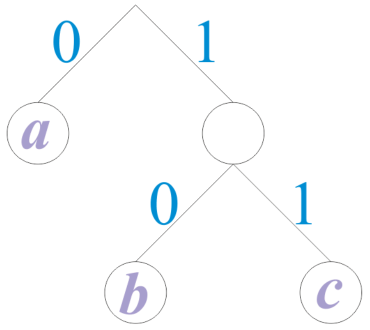
decoding speed MB/s/core (CPU):	2013	2021 year
Huffman	~300	~1000 MB/s
arithmetic	~50	~150 MB/s
ANS	-	~1500 MB/s GPU: 100+ GB/s

symbol sequence
complex probabilities

abaaabaabaa $\longleftrightarrow$ 0100101

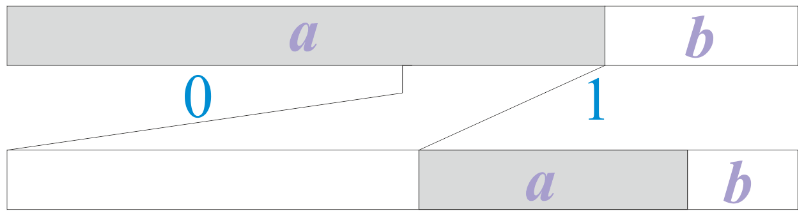
bit sequence
Pr(0)=Pr(1)=1/2

Past: compromise



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

or ?



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

Now: ANS

tANS: **tabled** - no multiplication
"Huffman generalized to fractional bits"
also allows for simultaneus encryption

rANS: **range** - direct replacement
of arithmetic/range coding: with
smaller state, less multiplications

mainly used for
smaller models,
fixed
distributions

fast (> 500MB/s/core)
uses nearly **accurate** Pr(*s*)
e.g. for Pr(*a*)=0.01, Pr(*b*)=0.99
uses ~**0.08** bits/symbol

mainly used for
larger models,
adaptive
distributions

10GB large **text** benchmark (2020, i9 9900K), 1GB wiki for 10 languages (ANS):

10GB -> Size	encoding time	decoding time	
5,034,758,325 bytes,	18.449 sec. -	7.311 sec.,	lz4 -1 (v1.9.2)
4,666,386,317 bytes,	26.686 sec. -	4.827 sec.,	lzturbo -10 -p0 (v1.2)
4,371,496,854 bytes,	46.907 sec. -	7.282 sec.,	lz4x -1 (v1.60)
3,909,521,247 bytes,	32.603 sec. -	11.287 sec.,	lizard -40 (v1.0.0)
3,823,273,187 bytes,	136.146 sec. -	59.070 sec.,	gzip -1 (v1.3.12)
3,770,151,519 bytes,	34.216 sec. -	26.236 sec.,	brotli -q 0 (v1.0.7)
3,642,089,943 bytes,	28.752 sec. -	10.717 sec.,	**zstd** -1 (v1.4.5) LZ + **tANS**/huf
3,660,882,443 bytes,	767.399 sec. -	7.633 sec.,	lz4x -9 (v1.60)
3,237,812,198 bytes,	392.835 sec. -	53.771 sec.,	gzip -9 (v1.3.12)
3,095,248,795 bytes,	137.881 sec. -	20.738 sec.,	brotli -q 4 (v1.0.7)
3,078,914,611 bytes,	240.124 sec. -	9.381 sec.,	**zhuff** -c2 -t1 (v0.99beta), LZ4 + **tANS**
3,065,081,662 bytes,	50.724 sec. -	12.904 sec.,	**zstd** -4 --ultra --single-thread (v1.4.5)
2,660,370,879 bytes,	153.103 sec. -	19.993 sec.,	**lzturbo** -32 -p0 (v1.2), LZ + **tANS**
2,639,230,515 bytes,	561.791 sec. -	11.774 sec.,	**zstd** -12 --ultra --single-thread(v1.4.5)
2,357,818,671 bytes,	3,953.092 sec. -	34.300 sec.,	rar -m5 -ma5 -mt1 (v5.80)
2,337,506,087 bytes,	2,411.038 sec. -	11.971 sec.,	**zstd** -18 --ultra --single-thread(v1.4.5)
2,220,027,943 bytes,	7,439.064 sec. -	22.690 sec.,	brotli -q 10 (v1.0.7)
2,080,479,075 bytes,	4,568.550 sec. -	12.934 sec.,	zstd -22 --ultra --single-thread(v1.4.5)
2,059,053,547 bytes,	4,909.124 sec. -	55.188 sec.,	7z -t7z -mx9 -mmt1 (v19.02) - LZMA
1,973,568,508 bytes,	6,626.946 sec. -	89.762 sec.,	arc -m9 -mt1 (v0.67)
1,921,561,064 bytes,	17,200.759 sec. -	27.147 sec.,	brotli -q 11 --large_window=30 (v1.0.7)
1,899,403,918 bytes,	1,327.809 sec. -	375.295 sec.,	nz -cO -t1 (v0.09 alpha)
1,722,407,658 bytes,	778.796 sec. -	401.317 sec.,	m99 -b1000000000 -t1 (beta)
1,675,874,699 bytes,	781.839 sec. -	198.309 sec.,	bwtturbo -59 -t0 (v20.2)
1,644,097,084 bytes,	21,097.196 sec. -	93.130 sec.,	**razor** (v1.03.7) - **adaptive 4bit rANS**
1,638,441,156 bytes,	1,030.489 sec. -	640.502 sec.,	bsc -m0 -b1024 -e2 -T (v3.1.0)
1,632,628,624 bytes,	1,146.133 sec. -	1,284.451 sec.,	bcm -9 (v1.40)
1,450,364,034 bytes,	2,701.335 sec. -	2,433.988 sec.,	mcm -x -m11 (v0.83)

***x* state stores information**

binary system: to encode symbol s:
rule: ‚new state' = 2• ‚old state' + s

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

example state: $1 \xrightarrow{s=0} 2 \xrightarrow{1} 5 \xrightarrow{1} 11 \xrightarrow{1} 23 \xrightarrow{1} 47$

asymmetric binary system (ANS):
rule: ‚new state' = number ‚old state' appearance of s

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

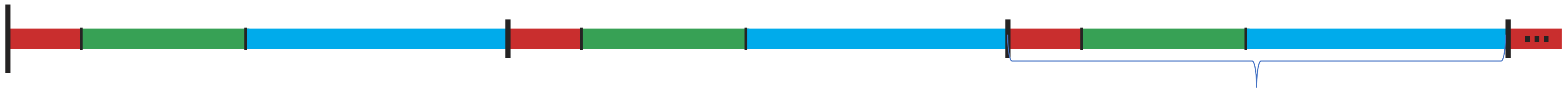
example stan: $1 \xrightarrow{s=0} 4 \xrightarrow{1} 6 \xrightarrow{1} 9 \xrightarrow{1} 13 \xrightarrow{1} 18$

symbol sequence „01111", encoded by binary system as 47,
ANS as shoorter (cheaper to write) 18
thanks to better agreement with: 1 more frequent than 0

ANS: $x \to \approx x/\Pr(s)$ while encoding symbol s

Redefine even/odd subsets according to densities

$x \to$ x-th appearance of 'even' ($s = 0$) or 'odd' ($s = 1$)



rANS variants: repeating division in ranges, e.g. of size 4:

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

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

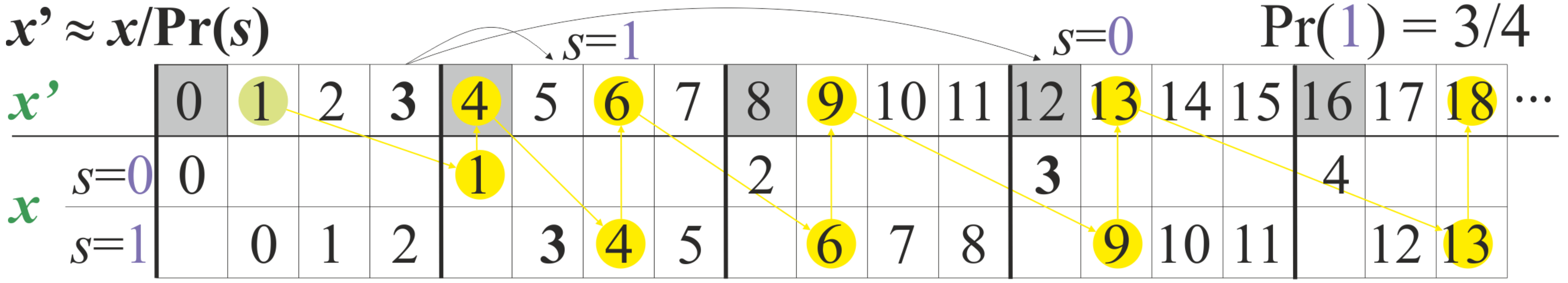
$$\text{if } \bar{s}(x) = 0,\ \ D(x) = (0, \lfloor x/4 \rfloor) \quad \text{else} \quad D(x) = (1, 3\lfloor x/4 \rfloor + \mathrm{mod}(x,4) - 1)$$

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

$\Pr(0) = 1/4$

$\Pr(1) = 3/4$

$x' \approx x/\Pr(s)$



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

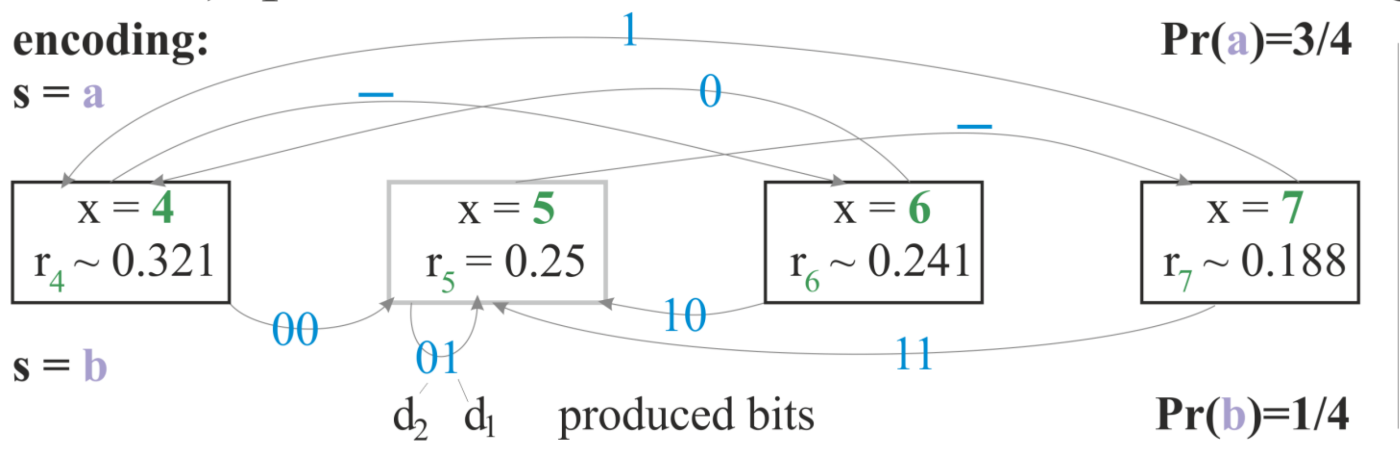
e.g. $x = 1 \xrightarrow{s=0} 4 \xrightarrow{1} 6 \xrightarrow{1} 9 \xrightarrow{1} 13 \xrightarrow{1} 18$

+ **renormalization** – make $x \in I$ e.g. $\boldsymbol{I=\{4,5,6,7\}}$ **below**, $\boldsymbol{I = [2^{16}, 2^{32}-1]}$ **rANS**

tANS (**tabled**, 2007): put into table with renormalization, building **automaton**

example:
binary
alphabet
$x \in \{4, 5, 6, 7\}$
states

$\Pr(a) > 1/2$
a **carries**
< 1 bit



decoding:

$x \to s$, new x

$4 \to a$, $6+d_1$

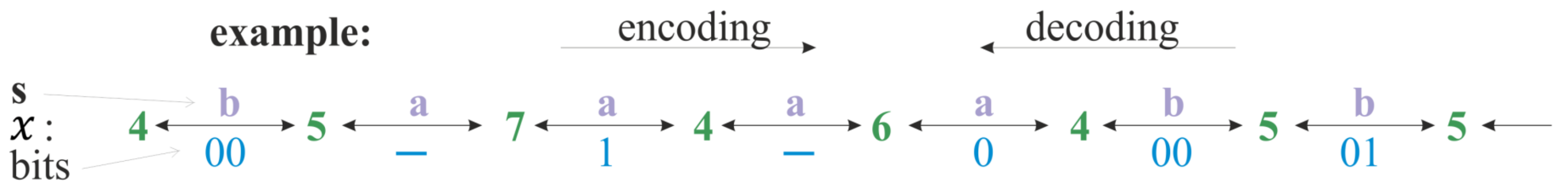
$5 \to b$, $4+2d_2+d_1$

$6 \to a$, 4

$7 \to a$, 5

$\underbrace{\qquad}_{\text{newX}} \; \underbrace{\qquad}_{\text{nbBits}}$

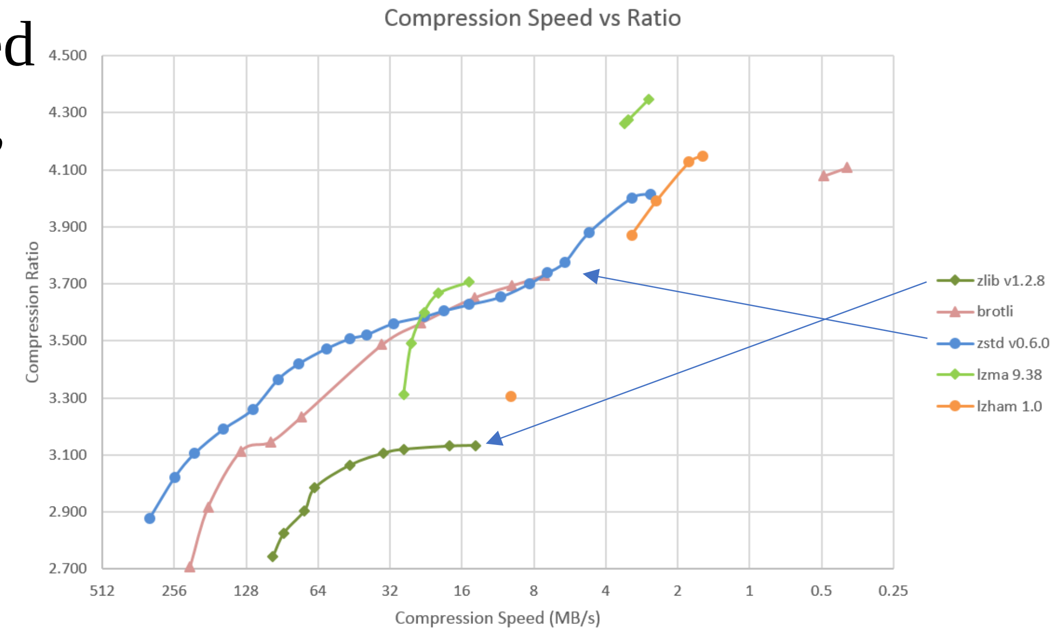
decodingTable

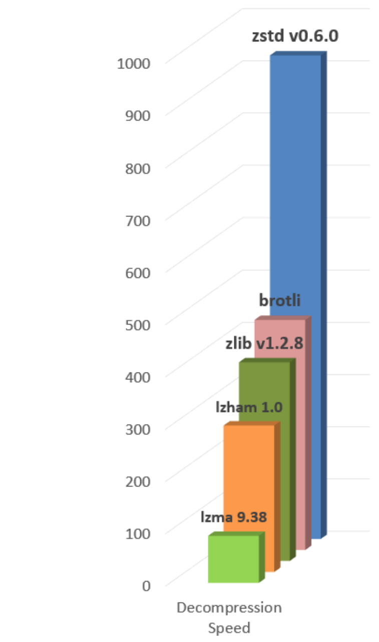


tANS used e.g. as FSE – Finite State Entropy (Yann Collet)

(gzip→) **in Zstd** – widely used
e.g. Facebook, **Linux kernel**,
lots of software, corporations

Apple LZFSE – default
in iPhone, Mac





tANS (2007) - **fully tabled behavior** for given probability distribution

Apple LZFSE, **Facebook ZSTD**, **lzturbo** … "**Huffman + fractional bits**"

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 (CSPRNG 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 one multiplication per symbol, good for SIMD/GPU

CRAM (DNA), **RAZOR**, **BB-ANS**(neural networks), **JPEG XL**, **GPU** (100+ GB/s)

Works directly on probabilities – more flexible, adaptivity

more memory effective – especially for large alphabet and precision, Markov

rANS decoding step ($mask = 2^n - 1$)	**Encoding step** (s) ($msk = 2^{16} - 1$, d = 32-n)
s = **symbol**(x & $mask$); writeSymbol(s);	if($x \geq (f[s] << \mathrm{d})$)
$x = f[s]\ (x >> n) + (x\ \&\ mask) - c[s]$;	{write16bits(x & msk); x >>= 16; }
if($x < 2^{16}$) $x = x << 16$ + read16bits();	$x = \lfloor x / f[s] \rfloor << n + (x \% f[s]) + c[s]$;

MB/s: **tANS/FSE: 380/500** **rANS**: **500/1500** … **GPU** rANS: **100+ GB/s**

Arithmetic coding (**AC**) **1976** … **lots of patents** … widely used in h.264 (**2004**)

Basic **timeline** of (AC→)**ANS**: **large community just sharing own work**:

2006 – first ANS variant in my physics MSc thesis (translation + later tANS)

2007,8 – tANS variant, implementations by Matt Mahoney, Andrew Polar

2013: Yann Collet tANS/FSE/zhuff, my often cited paper later introducing rANS
2014: Fabian Giesen rANS, James Bonfield very fast **Markov rANS** + CRAM

2015: Zstandard later Facebook, **Adaptive rANS**, Apple LZFSE
and maaany more: https://encode.su/threads/2078-List-of-Asymmetric-Numeral-Systems-implementations

… and the (known?) **patent attempts** – pursue of **monopoly**:

- 2015 Storleap **Markov rANS** – granted after restrictions (materials)

- **1.1.2014** I have written **Google** codec-devel mailing list to use ANS in video compression, helped them for 3.5 years counting for a formal collaboration …

June 2017 accidentally finding Google patent filled 1.5 years earlier

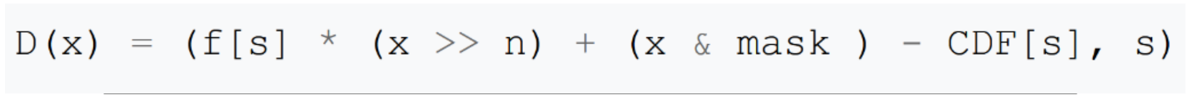
The patent described CABAC-like entropy coder for video: replacing AC with ANS, finally **rejected** – materials, Arstechnica, EFF

- **Microsoft**: I was aware for a year (materials), regularly checked Global Dossier which looked safe, but in 2022 the patent was **granted** by USPTO (The Register)

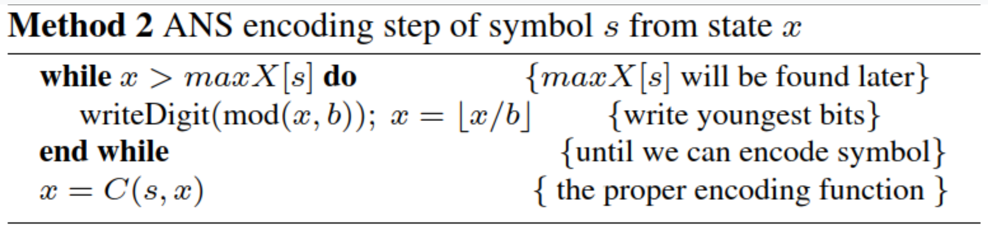
Can e.g. free software, Linux safely use JPEG XL which encodes with rANS?

Claim 1 (of 28): “A computer system comprising: an encoded data buffer configured to store encoded data from a bitstream; and a range asymmetric number system (“RANS”) decoder configured to perform operations using a **two-phase structure for RANS** decoding operations, the operations comprising: during a first phase of the two-phase structure, selectively updating, depending on a determination of whether or not an output symbol from a previous iteration was generated, state of the RANS decoder **using probability information** for the output symbol from the previous iteration, the state of the RANS decoder being tracked using a value; during a second phase of the two-phase structure, selectively **merging a portion of the encoded data from an input buffer into the state** of the RANS decoder; and during **the second phase of the two-phase structure, selectively generating, depending on a determination of whether** or not the state of the RANS decoder includes sufficient information to generate an output symbol for a current iteration, the output symbol for the current iteration using the state of the RANS decoder, the state of the RANS decoder including sufficient information to generate the output symbol for the current iteration if the state of the RANS decoder is greater than a **threshold**.”

Much earlier Wikipedia, article “two-phase”



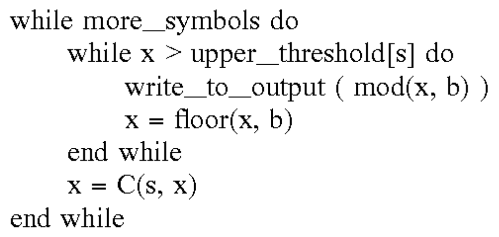
```
D(x) = (f[s] * (x >> n) + (x & mask ) - CDF[s], s)
```



```
Method 2 ANS encoding step of symbol $s$ from state $x$
  while $x > maxX[s]$ do                          {$maxX[s]$ will be found later}
      writeDigit($\mathrm{mod}(x,b)$); $x = \lfloor x/b \rfloor$      {write youngest bits}
  end while                                       {until we can encode symbol}
  $x = C(s,x)$                                      { the proper encoding function }
```

patent granted by USPTO

$$x = f_s * (x >> n) + (x \& \text{ mask}) - c_s$$



```
while more_symbols do
    while x > upper_threshold[s] do
        write_to_output ( mod(x, b) )
        x = floor(x, b)
    end while
    x = C(s, x)
end while
```

How to defend your work from patent vultures?

What if somebody just copy&paste your repository and try to patent it?

Nearly impossible to find out … if so, what can be done?

Patent officer in practice often **search only for prior art in patents** …

"Third-party preissuance submission" … but what if **monopoly** got granted?

Suggestion: maybe **some fundraising to get rid of inconvenient patents?**

For **separate patents** and general pool, guarantee of legal action if threshold …

Fundraising from individuals, but also **larger entities blocked by given patent**

e.g. rANS – used, developed e.g. by JPEG, Google, Facebook, NVIDIA …

By the way bringing **spotlight for the general pathology** … **discussion/change?**

How to organize it? Defend from being forbidden to use own work?

Can e.g. free software, Linux safely use JPEG XL which encodes with rANS?

See also http://datageekdom.blogspot.com/2018/09/mpeg-g-ugly.html

genetic compressor authors enforced to switch to full time patent defenders