



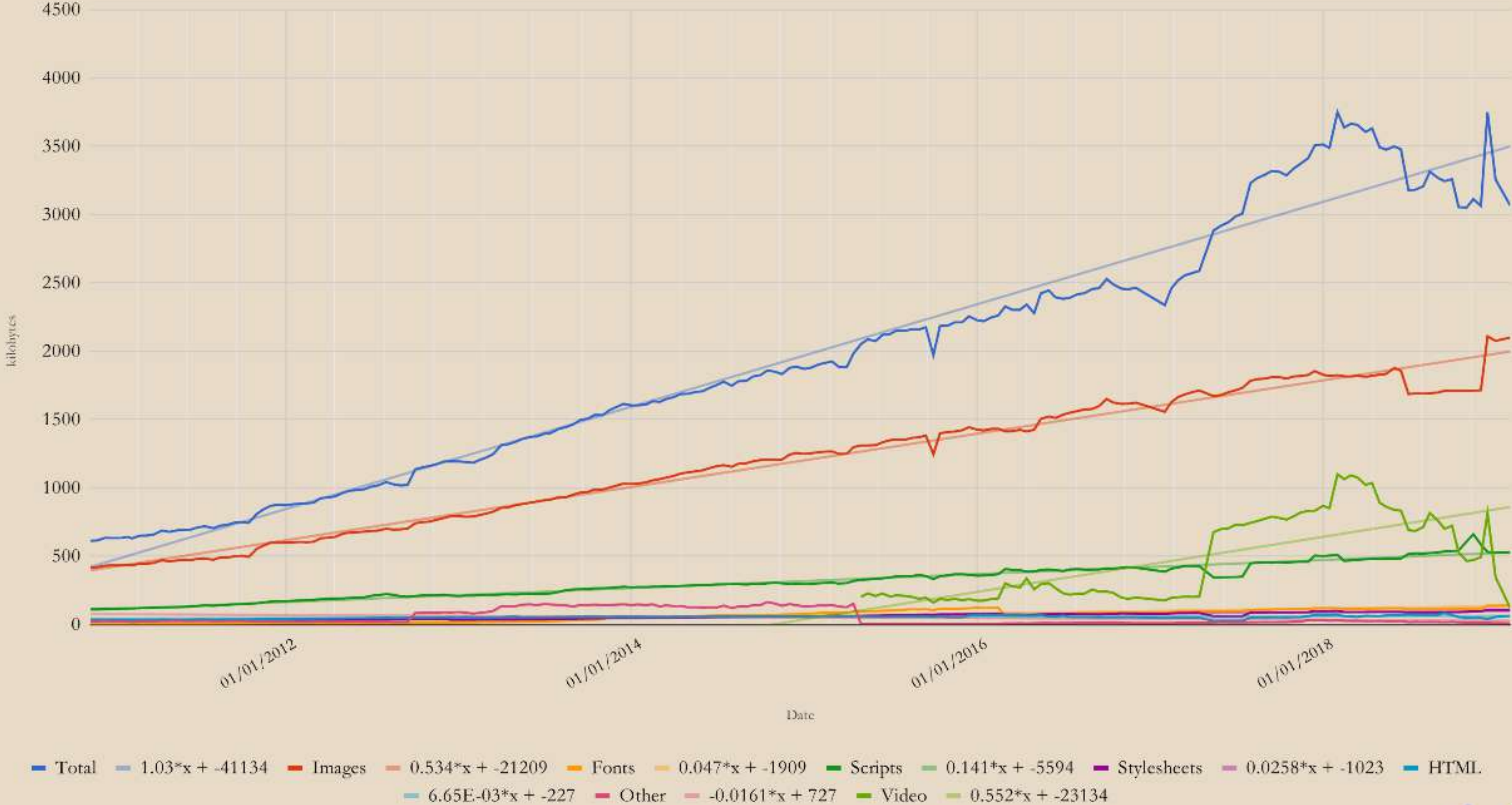
# **A deep dive into images on the web**

**and then some...**

Chen Hui Jing / @hj\_chen



Page Weight Chart





Surname First name

陈	慧	晶
Chen	Hui	Jing



@hj\_chen



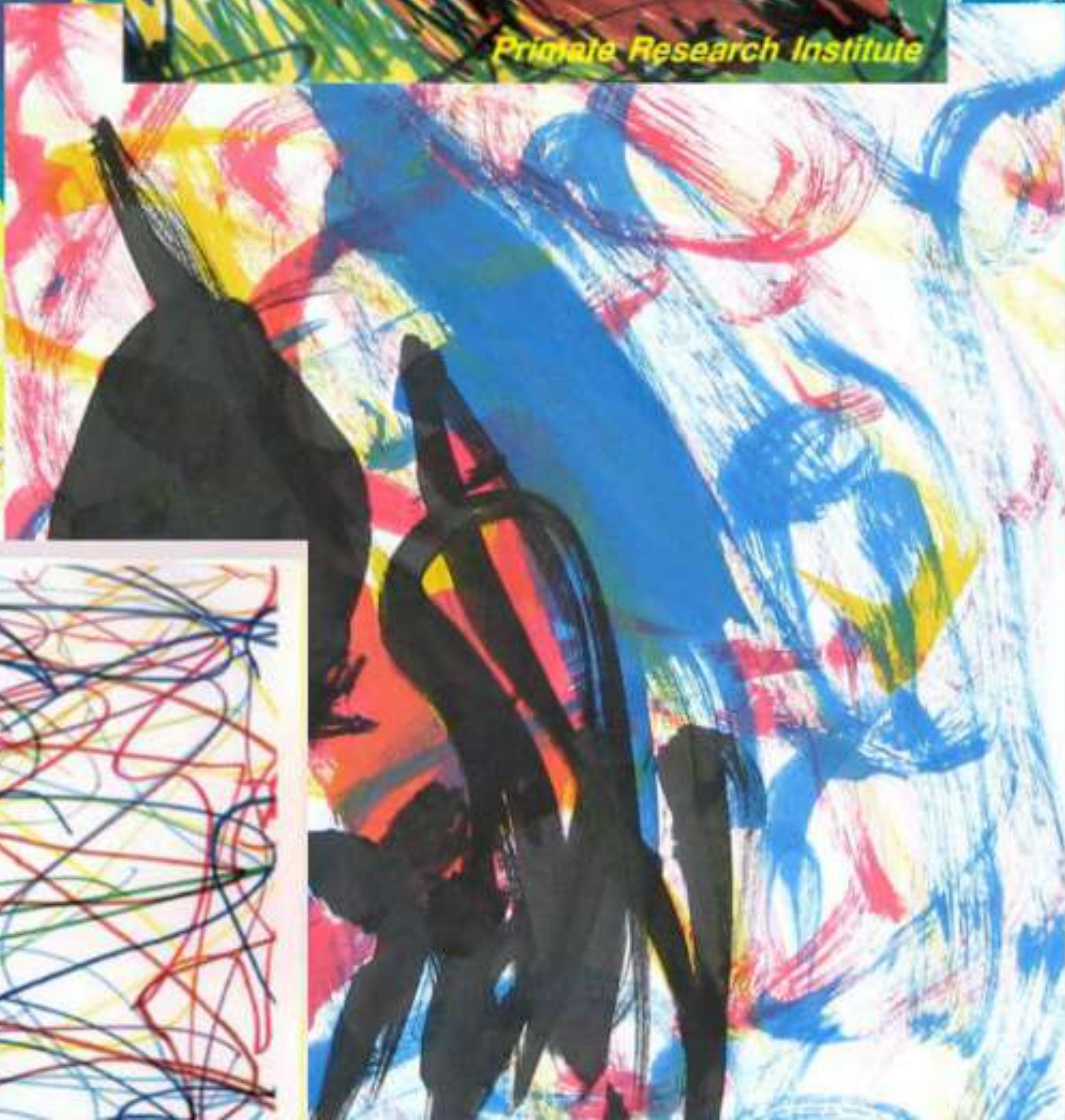




🥑 Developer Advocate 🥑

**nexmo**®  
The Vonage® API Platform









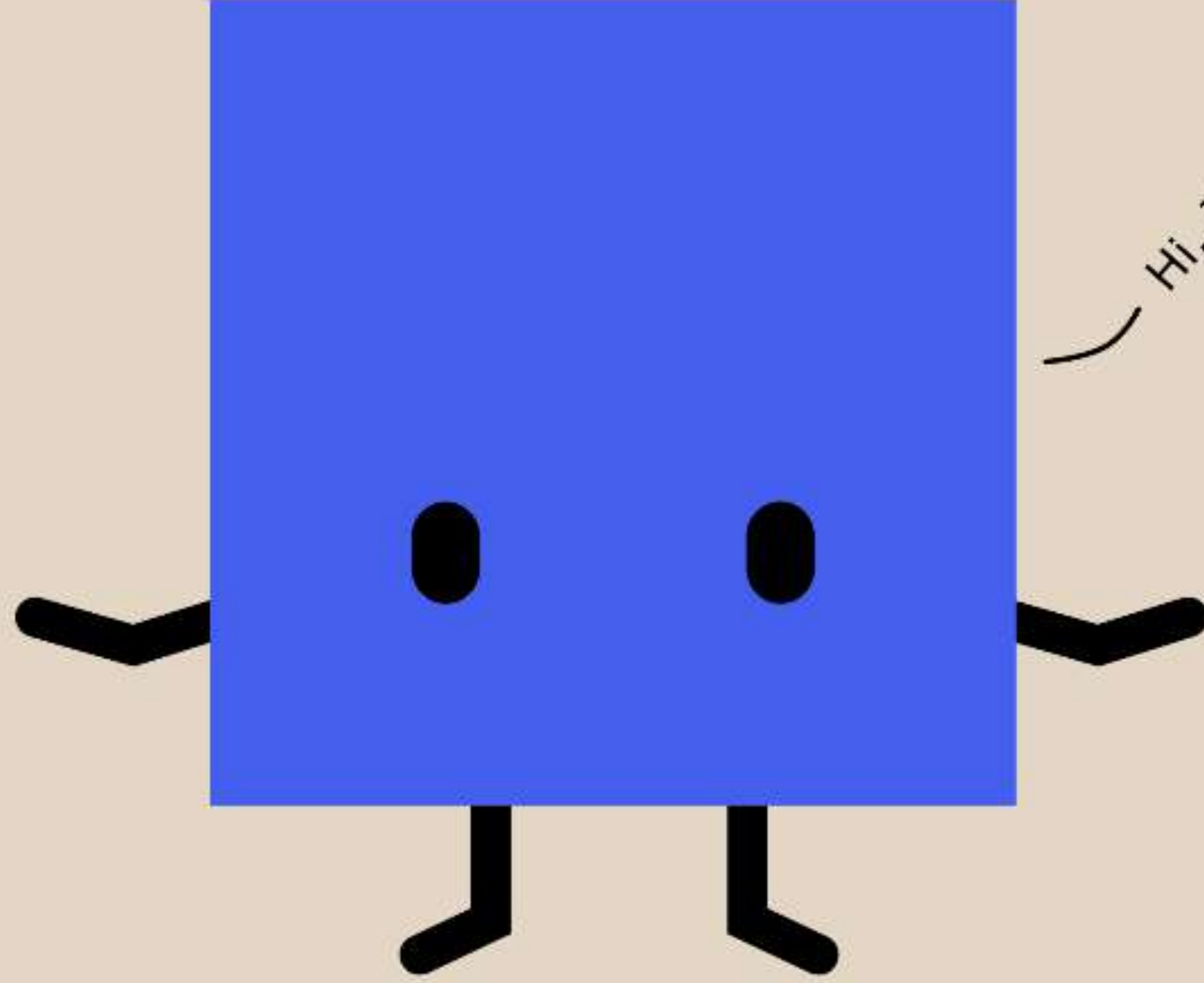






**ELECTRONIC SIGNALS**





Hi, I'm Pixel...



Picture



Pics

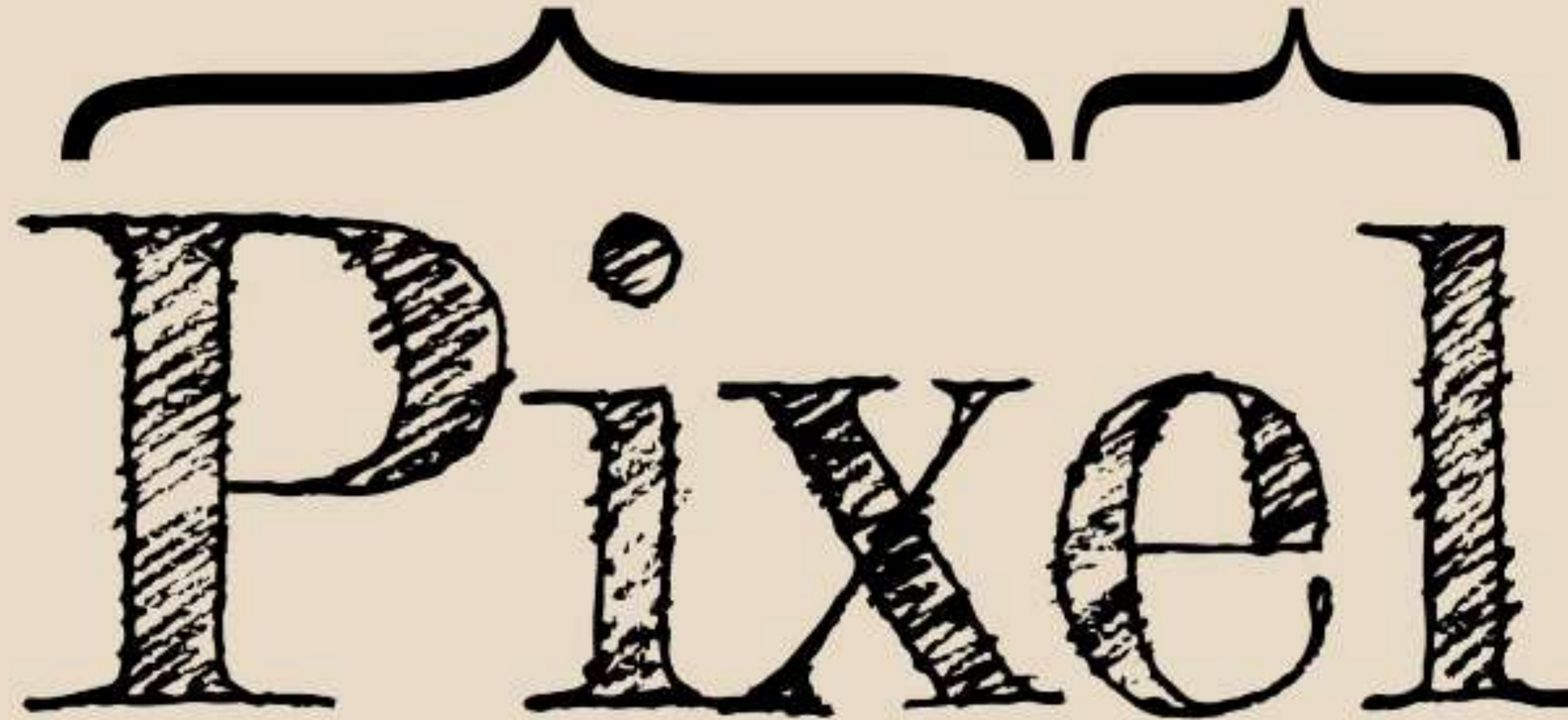


Pix

Element

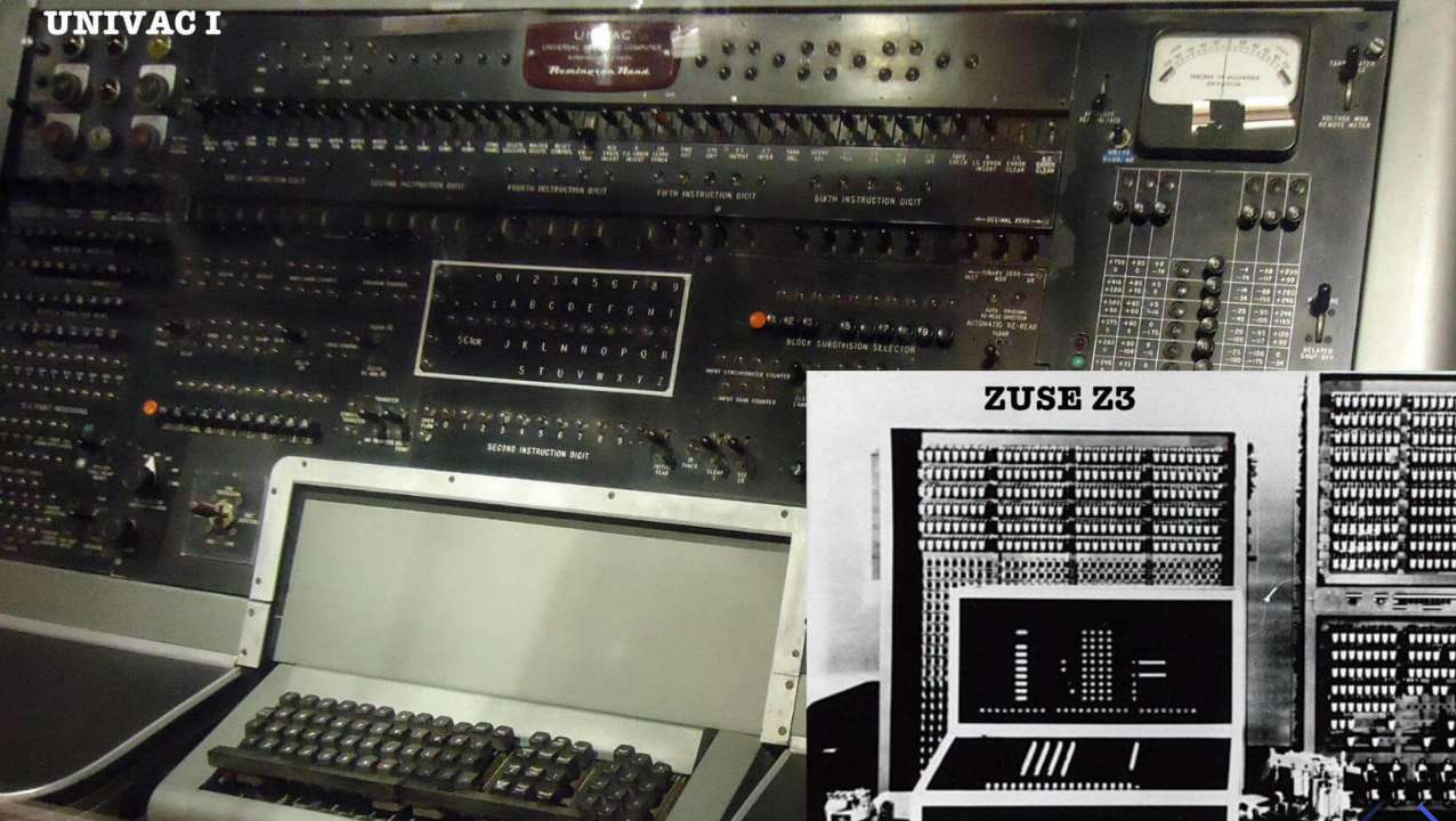


el

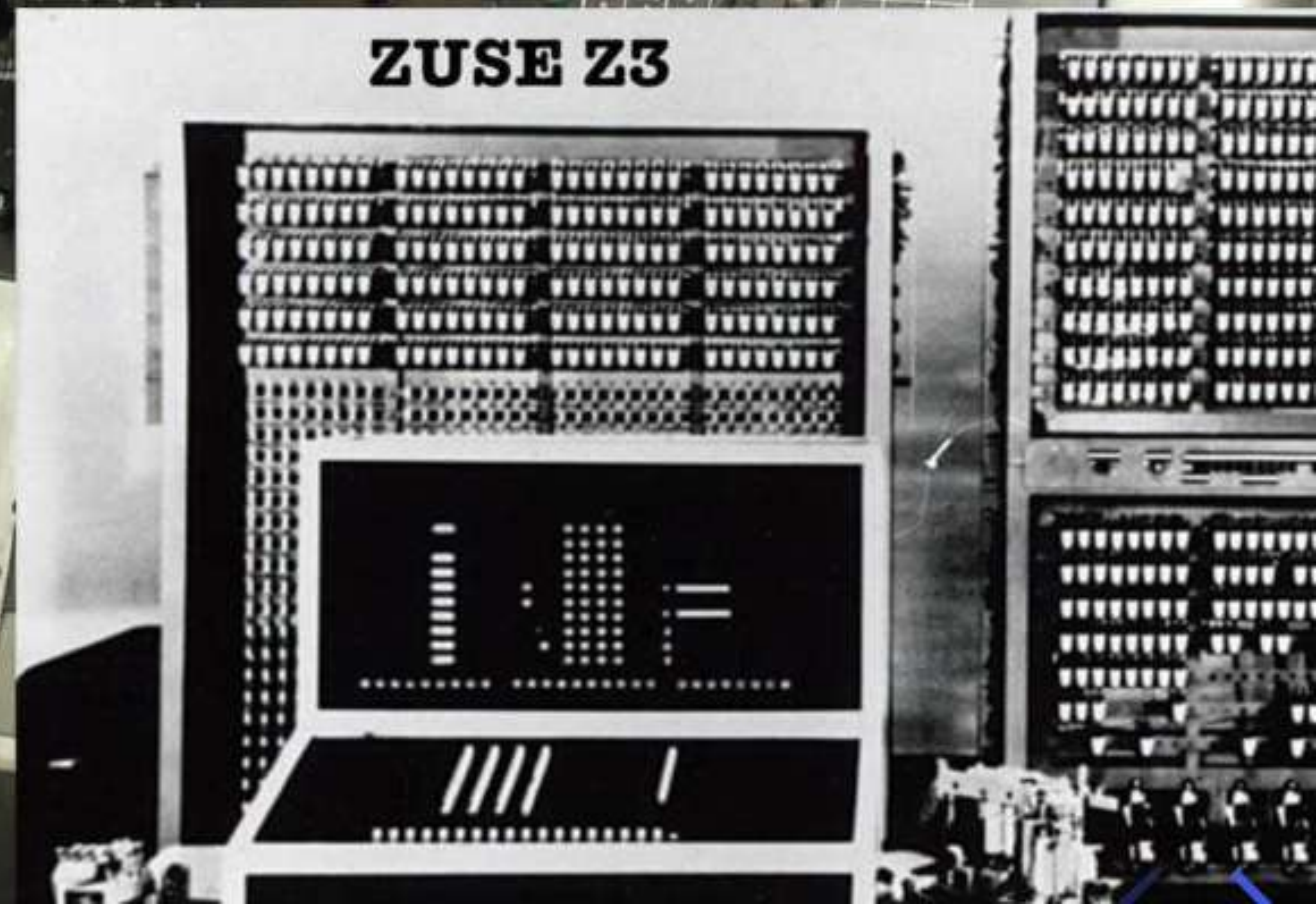
Pixel



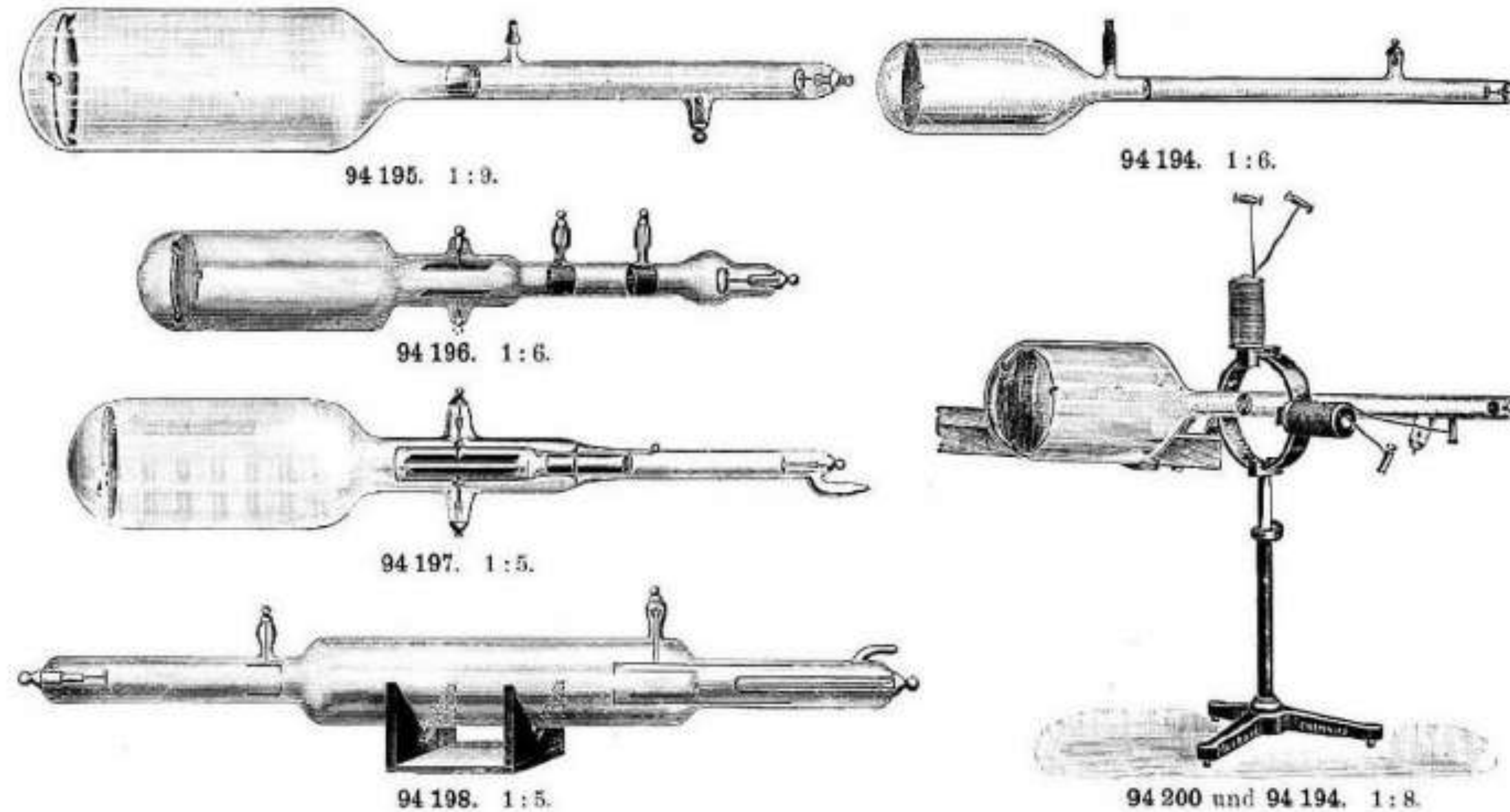
# UNIVAC I



# ZUSE Z3





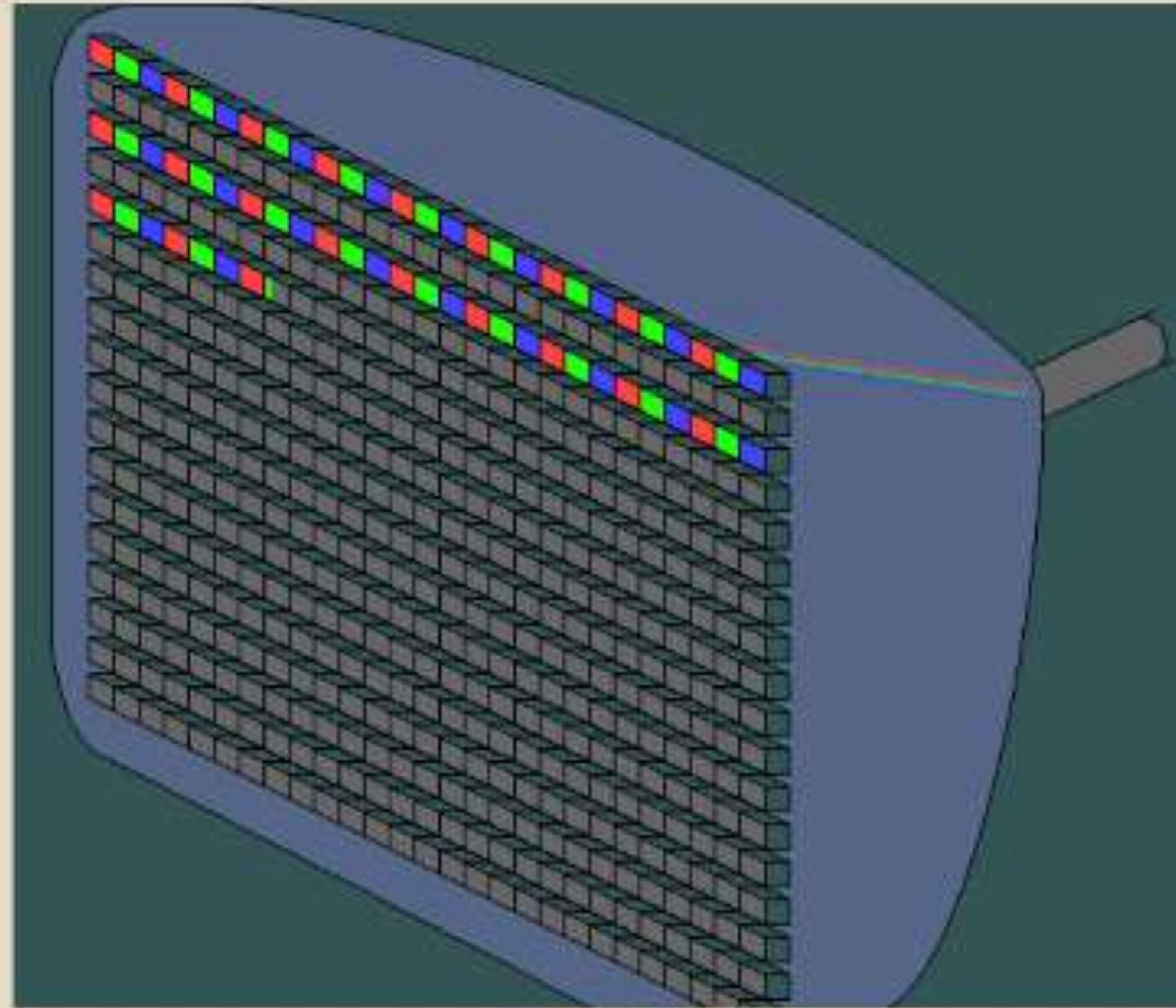


- 94 194. Kathodenstrahlenröhre nach Braun, Figur (Wied. Ann. 1897, Seite 552). Beschreibung  
 94 195. — dieselbe, größer, Figur, 1 m lang, mit Schirm von 130 mm Durchmesser  
 94 196. Vakuumröhre nach Thompson, Figur, zum Studium der Ablenkbarkeit der  
 Kathodenstrahlen durch statische Elektrizität und durch Magnet . . . . .  
 94 197. — desgl. nach Braun-Wehnelt, Figur, für elektrostatische Ablenkungen,  
 mit Faradayschem Käfig als Diaphragma . . . . .  
 94 198. Vakuumröhre nach Perrin, zur Vorführung der durch Kathodenstrahlen hervor-  
 gerufenen negativen Ladung, Figur, mit Fuß . . . . .  
 94 199. Universal-Stativ nach Simon und Reich, für Versuche mit der Braunschen  
 Kathodenstrahlenröhre, Figur . . . . .

CRTs sold by Müller-Uri, Source: [The Cathode Ray Tube site](#)



# Raster scanning



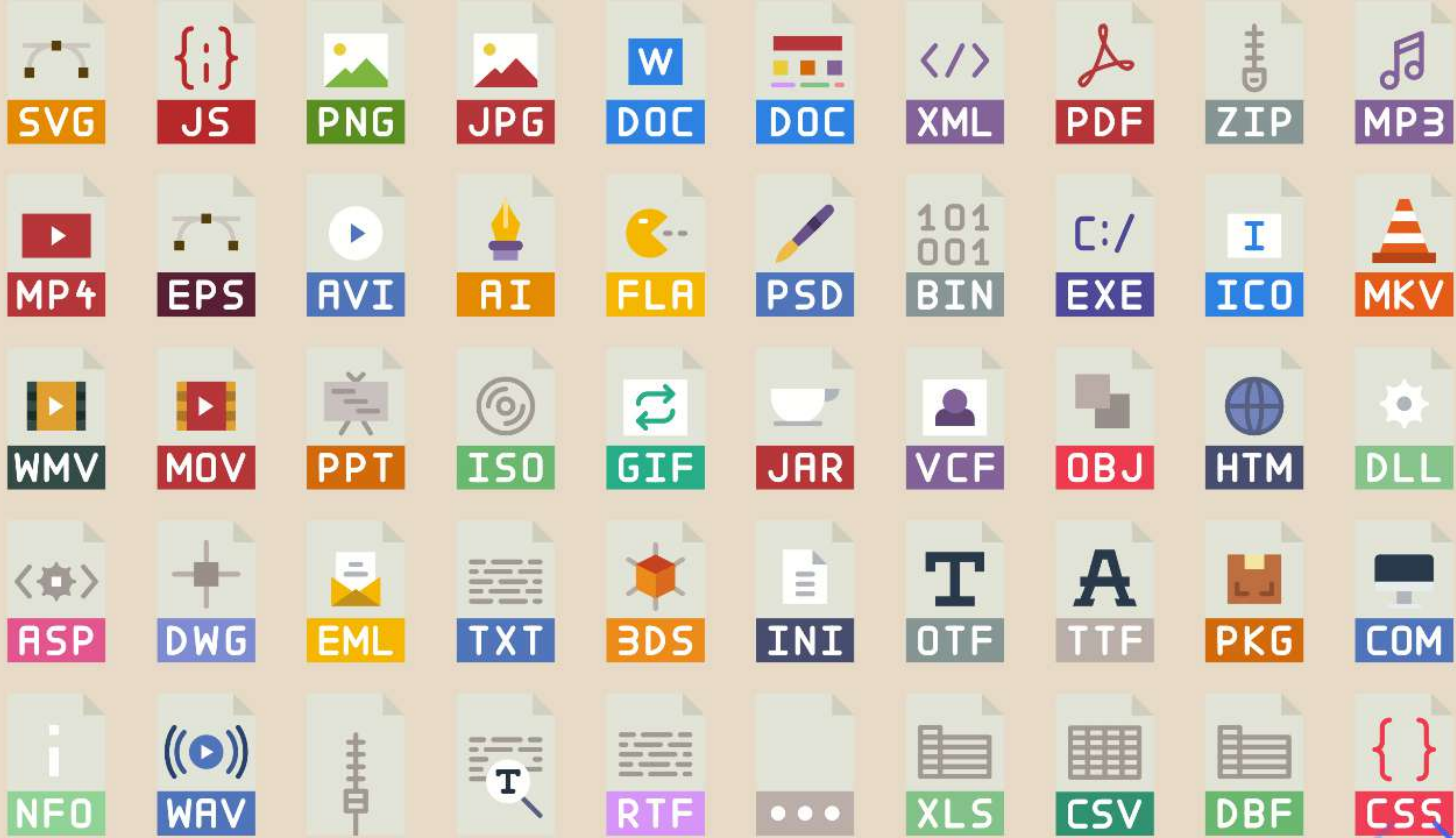
Source: [M-SYS MV](#)



	<b>Raster scan</b>	<b>Random scan</b>
Electron beam	Swept across entire screen, one row at a time, from top-to-bottom	Only directed to parts of the screen where image is drawn
Resolution	Poor, due to plotting as discrete point sets	Good, as CRT beam directly follows line path
Picture definition	Stored as set of intensity values (pixels) in refresh buffer area	Stored as set of line drawing instructions in display file
Realism	Variable intensity values allow for realistic shadow and colour patterns	Most suited for line drawing
Drawing method	Screen points (pixels)	Mathematical functions

Source: Prof. Vijay M. Shekhat, CE Department, [Computer Graphics](#)







# File formats

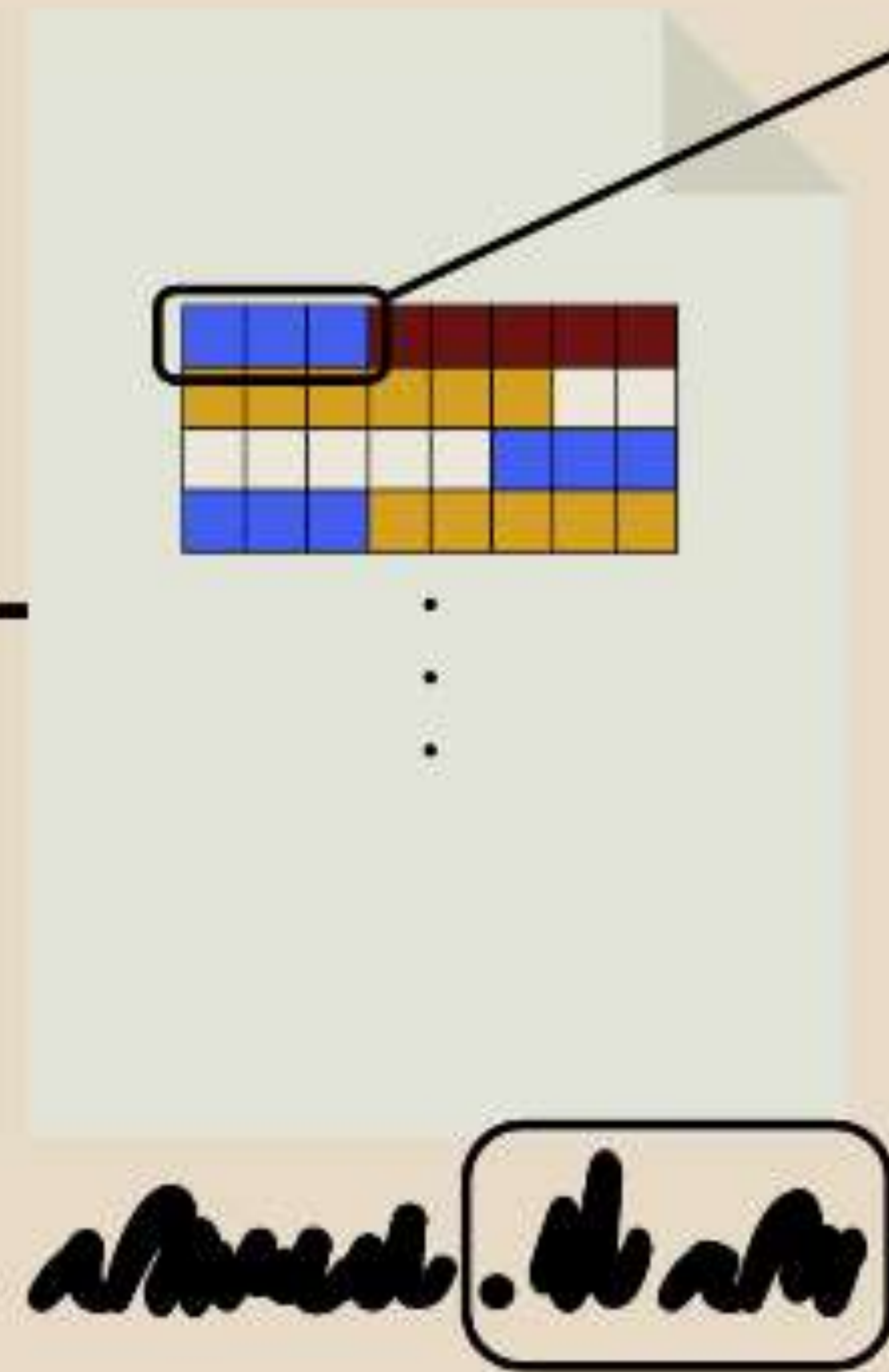
## External metadata

Mac OS type-codes  
Mac OS X UTIs  
OS/2 extended attr.  
POSIX extended attr.  
PUIDs  
MIME types  
FFIDs  
File content-based identification



## Internal metadata

File header / Magic number



File extension



# Colour depth



**1-bit PNG**  
(2 colours)



**2-bit PNG**  
(4 colours)



**4-bit PNG**  
(16 colours)



**8-bit PNG**  
(256 colours)



**24-bit PNG**  
(16,777,216 colours)

Source: [Wikipedia, Color depth](#)



Edited by:

David Baggett

Internet:

dmb@ai.mit.edu

(Please report errors or additions.)

Copyright (C) 1988 -- 1995 by David M. Baggett

word = 2 bytes

long = 4 bytes

palette = Hardware color palette, stored as 16 words. First word is color register zero (background), last word is color register 15. Each word has the form:

Bit: (MSB) **MacPaint** 01 00 (LSB)  
 --- -- --  
 02 B1 B0

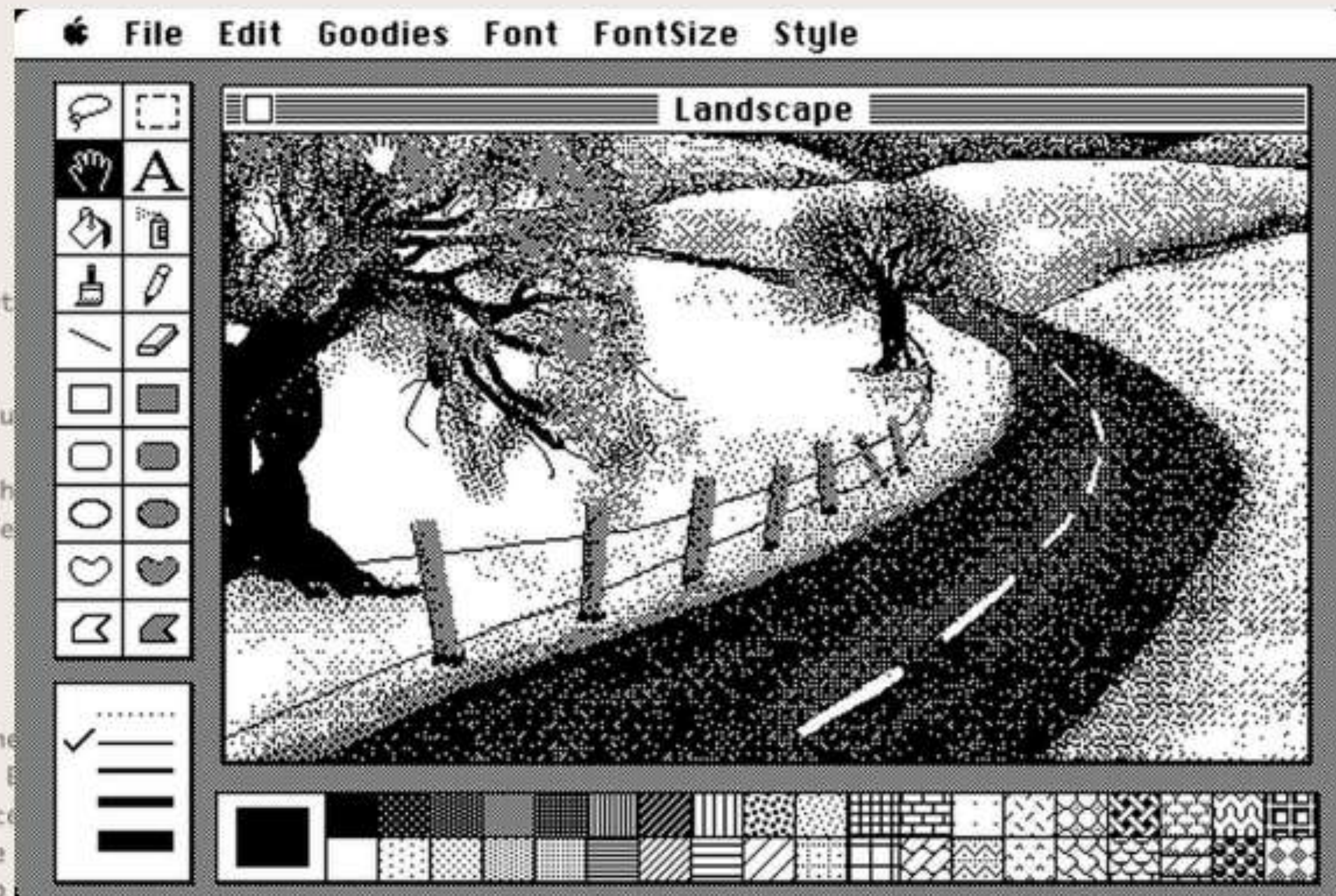
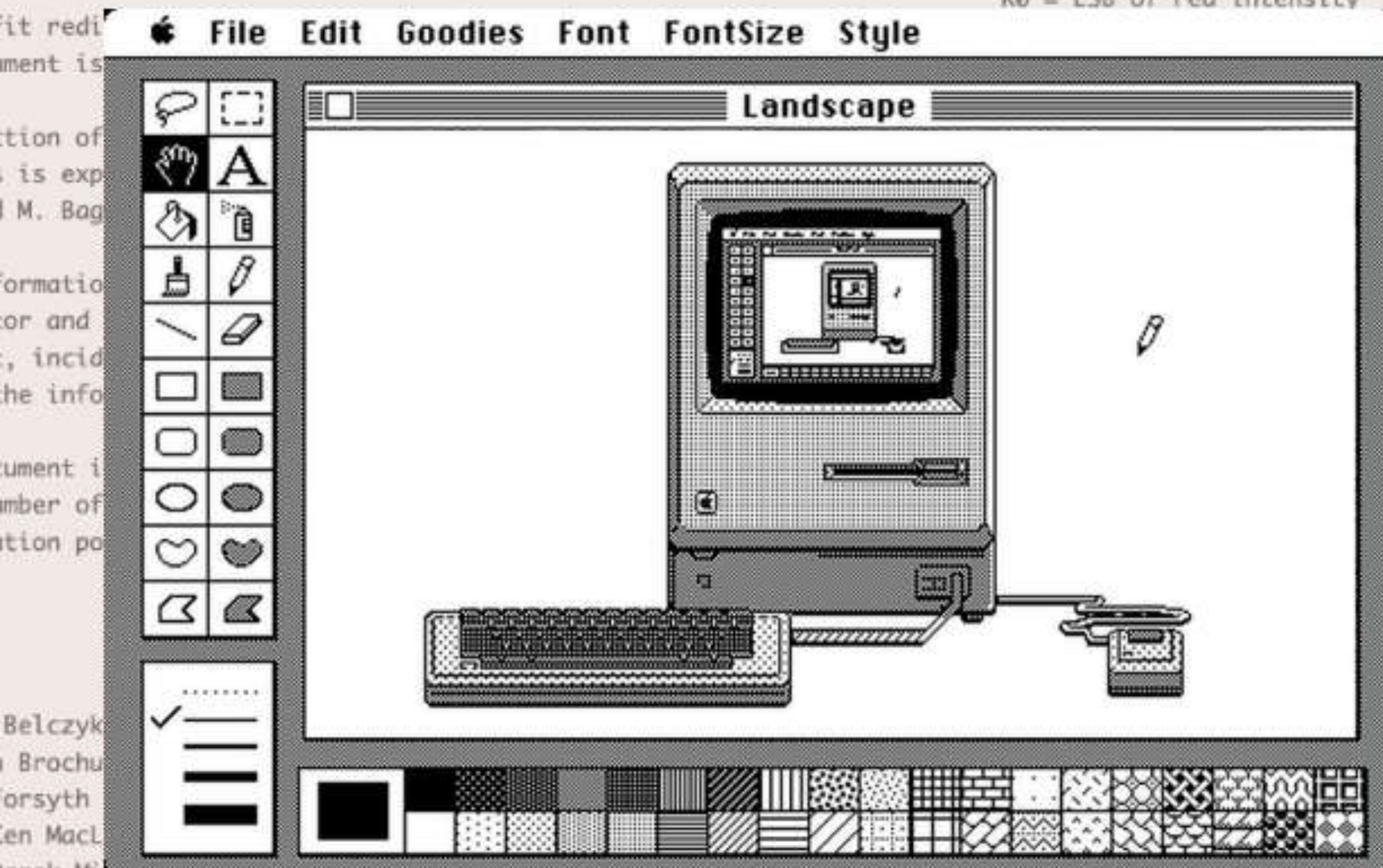
R2 = MSB of red intensity

R0 = LSB of red intensity

NOTE: The version number is actually a flag to MacPaint to indicate the brush/fill patterns are present in the file. If the version is 0, the default patterns are used. Therefore you can simply save a file by writing a blank header (512 \$00 bytes), followed by the image data.

Bitmap compression:

The bitmap data is for a 576 pixel by 720 pixel monochrome image. The packing method is PackBits (see below). There are 72 bytes per scan line. Each bit represents one pixel; 0 = white, 1 = black.



Chris Ridd George Seto Joe Smith Greg Wageman  
 Roland Walldi\* Gerry Wheeler

204 bytes unused

512 bytes total for header

&lt; 51200 bytes compressed bitmap data

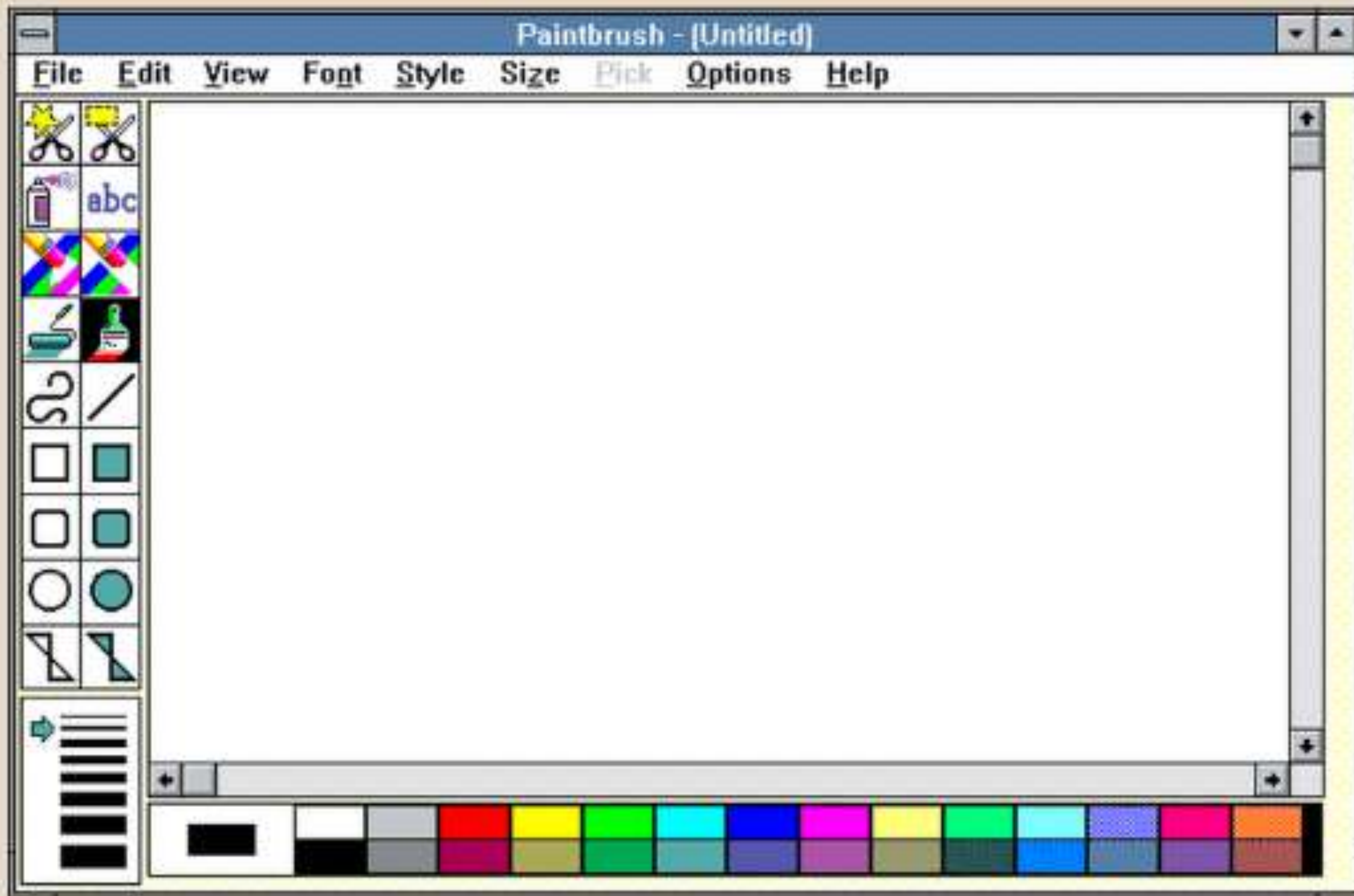
&lt; 51712 bytes total

GEM Bit Image, IFF, MacPaint

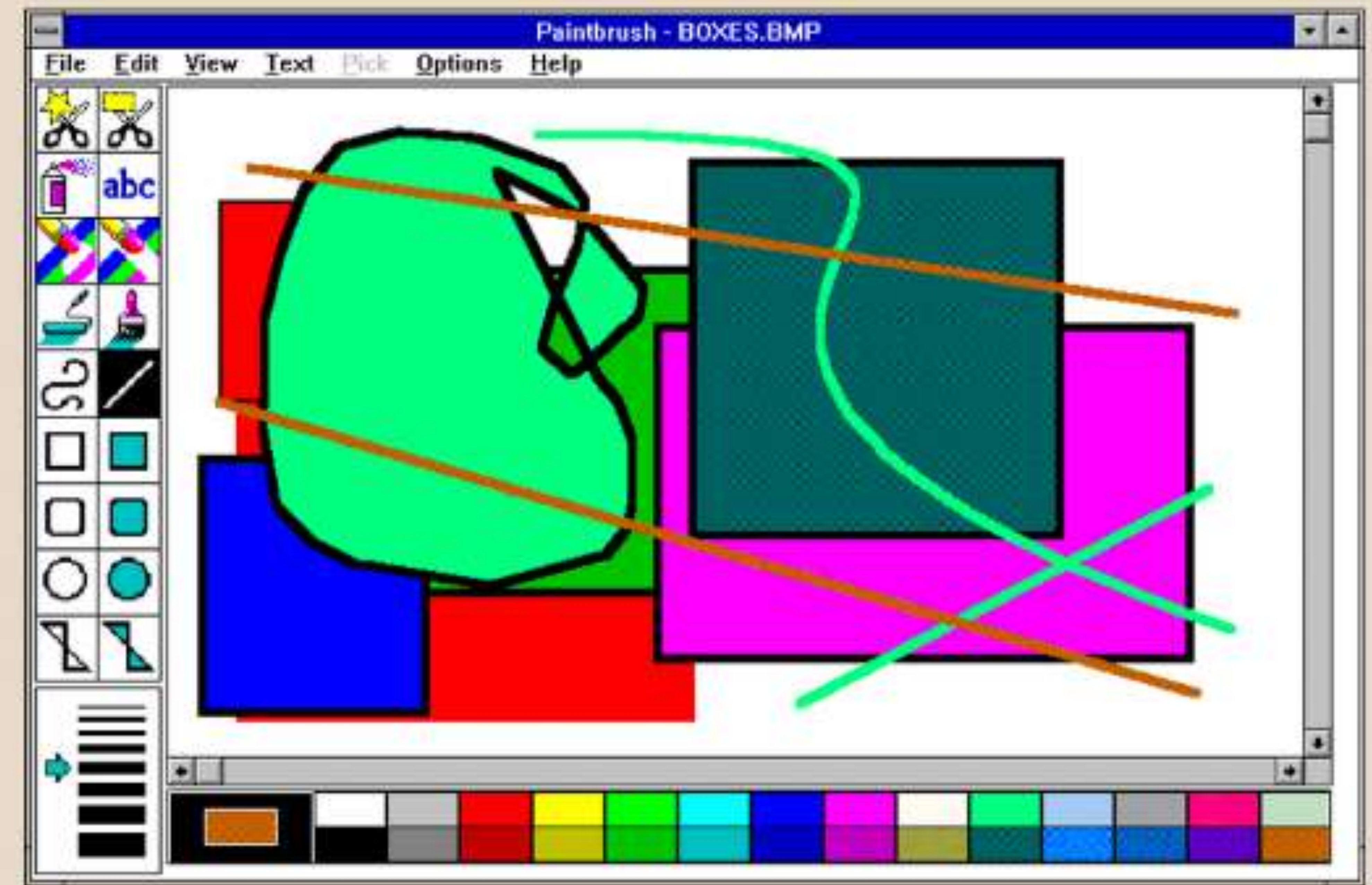
Version of.....Sun Oct 30 12:40:13 EST 1994  
 (Last change: Extended GEM .IMG format updated)



# MS Paintbrush



Paintbrush for Windows 3.0












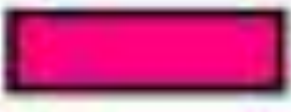


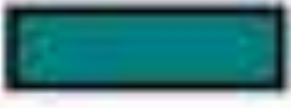



Paintbrush for Windows 3.1



# Bitmap (BMP)

3	3	3	3	3	3	3	3
0	1	4	1	4	1	4	0
0	4	1	4	1	4	1	0
0	5	5	5	5	5	5	0
0	5	5	5	5	5	5	0
0	1	4	1	4	1	4	0
0	4	1	4	1	4	1	0
2	2	2	2	2	2	2	2

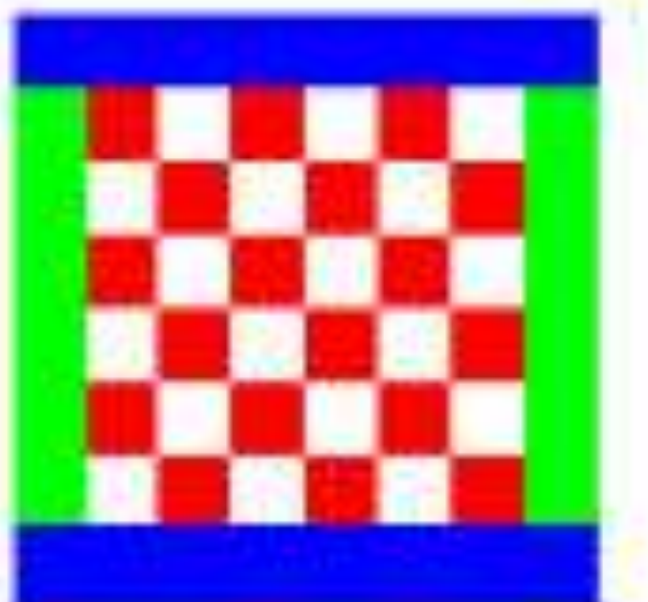
  

  

0	000000	
1	FF0000	
2	00FF00	
3	0000FF	
4	FFFFFF	
5	FFFF00	
6	FF00FF	
7	00FFFF	
8	FF0080	
9	FF8040	
A	804000	
B	008080	
C	800000	
D	800080	
E	8080FF	

```

0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
00FF00 FF0000 FFFFFFFF FF0000 FFFFFFFF FF0000 FFFFFFFF 00FF00
00FF00 FFFFFFFF FF0000 FFFFFFFF FF0000 FFFFFFFF FF0000 00FF00
00FF00 FF0000 FFFFFFFF FF0000 FFFFFFFF FF0000 FFFFFFFF 00FF00
00FF00 FFFFFFFF FF0000 FFFFFFFF FF0000 FFFFFFFF FF0000 00FF00
00FF00 FF0000 FFFFFFFF FF0000 FFFFFFFF FF0000 FFFFFFFF 00FF00
00FF00 FFFFFFFF FF0000 FFFFFFFF FF0000 FFFFFFFF FF0000 00FF00
0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF

```



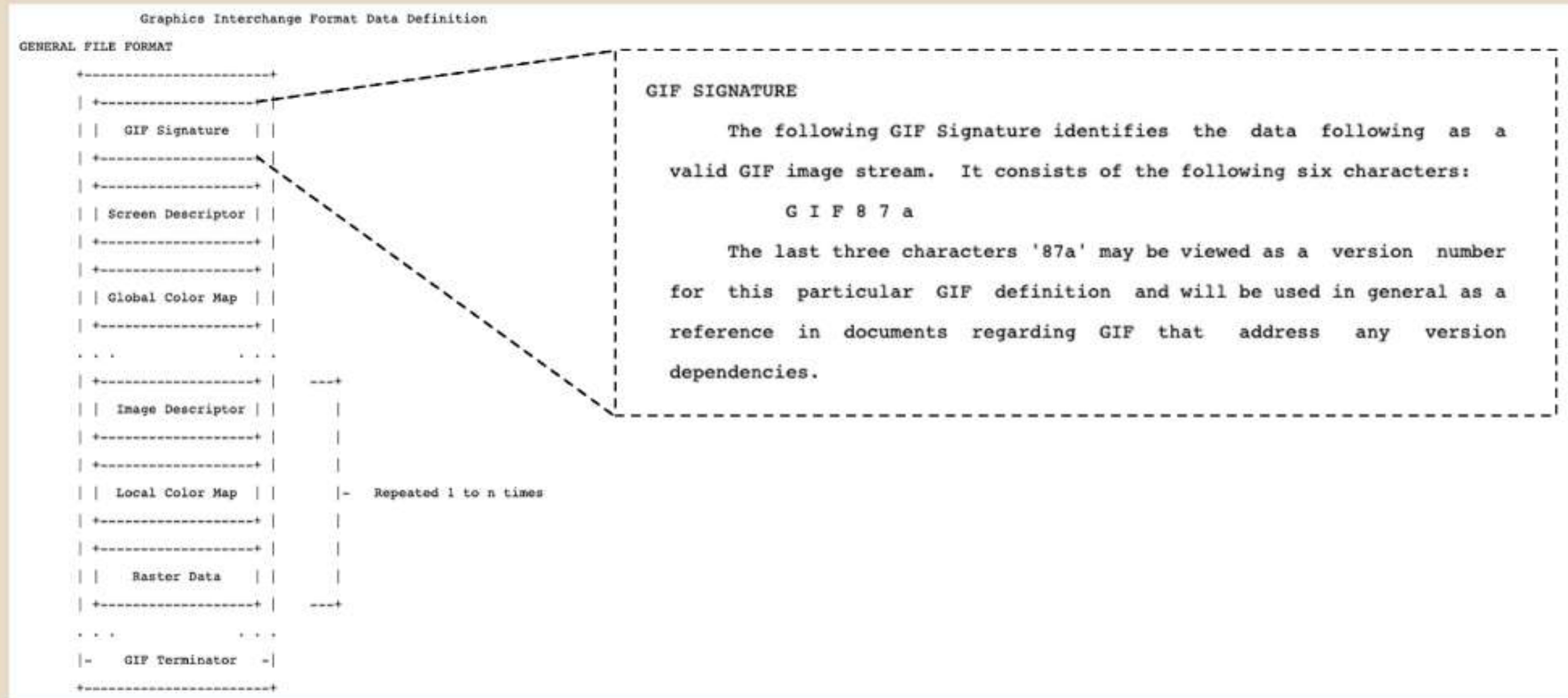
Colours directly in bitmap itself

Bitmap and its corresponding colour table

Source: [Microsoft, Types of Bitmaps](#)



# Graphics Interchange Format (GIF)



Source: [Graphics Interchange Format \(tm\)](#)



# Compression algorithms

Run-length Compression  
20 Bytes → 16 Bytes

B A B A B A B A B A B A 4 C 4 A

Run length compression used by MacPaint

LZW aka GIF Compression

Q = B A + Q Q Q Q Q Q C C C C A A A A  
4 Bytes 14 Bytes

Q = B A  
4 Bytes

Lempel-Ziv-Welch (LZW) compression

Motion graphics by [Crystal Law](#)





First band photo on the web

Source: [The Cernettes](#)



# Joint Photographic Experts Group (JPEG)





# JPG compression



Source RGB image



Colourspace conversion

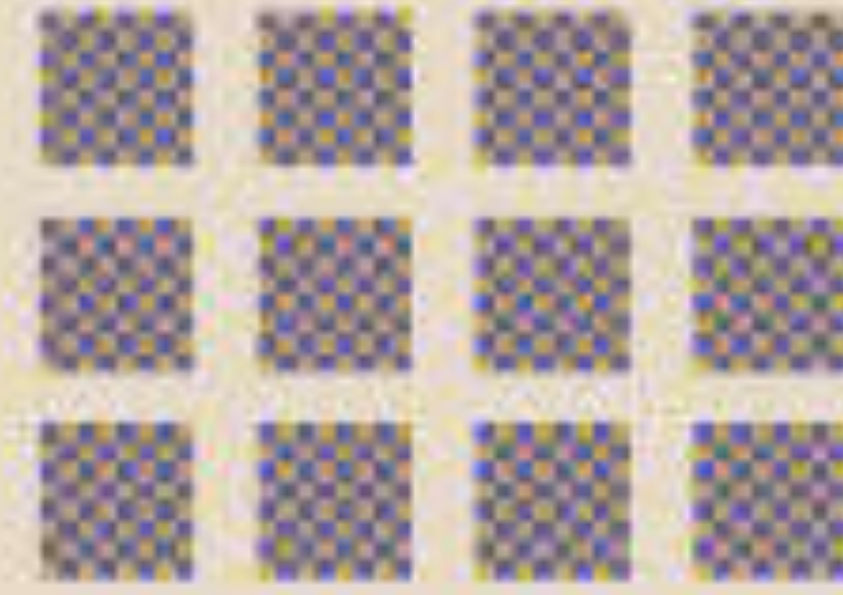


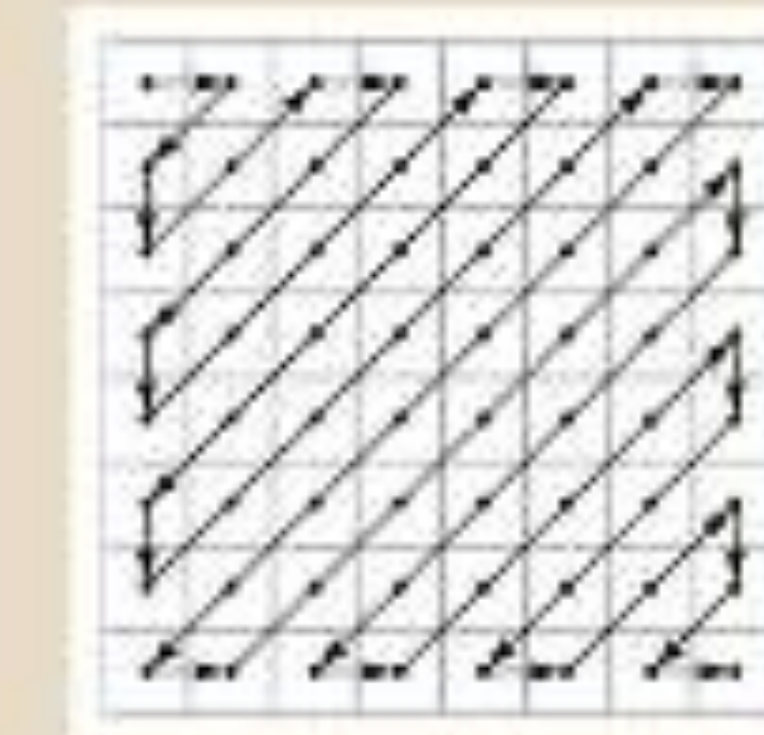
Image split into blocks

52 55 61 66 70 63 64 73							
63 59 55 50 109 95 89 72							
62 95 60 113 144 104 86 73							
63 98 94 433 483 482 498 601	-64 -55						
67 61 65 488 428 386 420 161	-59 -56						
79 55 66 89 87 78 70 75	62 -56						
85 72 63 56 48 38 35 35	205 225 94	56.12	-20.10	-2.39	0.46		
87 75 63 63 48 34 34 34	75 60 125 85	13.15	-7.09	-0.54	4.80		
-49 -84 -88 -58 -51	128 178 248 36	-28.51	9.53	5.42	-5.85		
-43 -98 -88 -69 -71	167 176 144 96	-10.24	6.90	1.83	1.99		
-41 -163 -88 -60 -54	-103 200 -138 5	-1.87	1.76	-2.39	1.14		
-1.73	2.01	2.38	-5.94	-2.10	0.94	4.10	1.85
-1.03	0.10	0.42	-2.42	-0.88	-3.02	4.12	-0.86
-0.17	0.14	-1.07	-4.19	-1.17	-0.10	0.80	1.68

Forward Discrete Cosine Transform

16 13 10 14 34 40 51 61							
12 13 14 19 36 50 60 55							
14 13 16 24 40 57 69 56							
14 17 22 29 51 67 80 82							
18 22 27 36 68 60 103 102	-170 0						
26 26 36 60 81 104 112 82	0 0						
49 64 70 82 103 111 128 101	0 0						
12 92 25 58 132 100 100 0	0 0						
0 0 0 0 0 0 0 0							
0 0 0 0 0 0 0 0							
0 0 0 0 0 0 0 0							

Quantisation



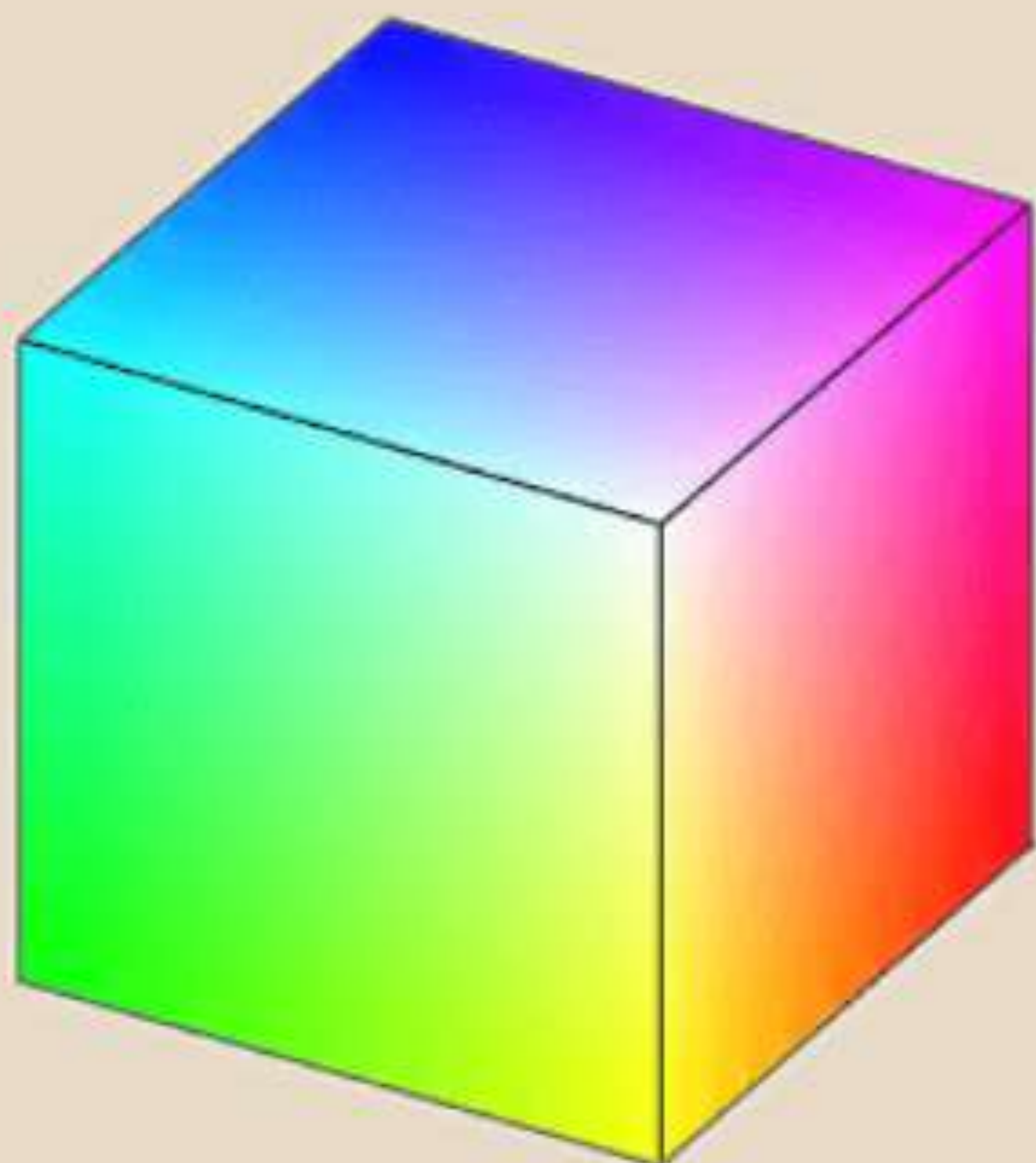
Statistical encoding



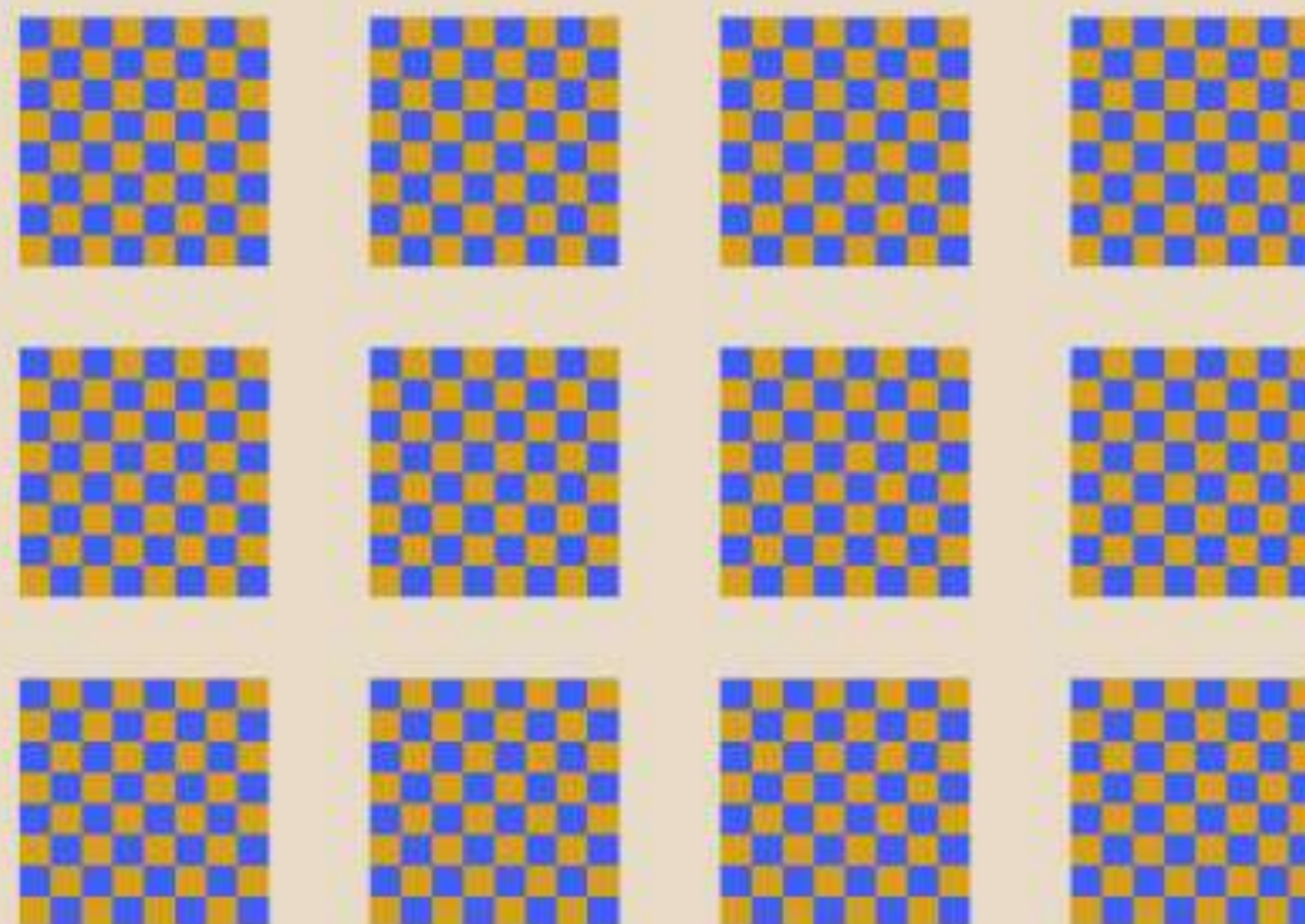
.JPG file

Reference: [How JPG works](#)





**Colourspace conversion**



**Image split into blocks**

Reference: [How JPG works](#)



16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	82
49	64	78	83	103	121	120	101
72	92	95	98	112	100	103	89
			0	0	0	0	0
			0	0	0	0	0
			0	0	0	0	0

Quantisation

Reference: [How JPG works](#)

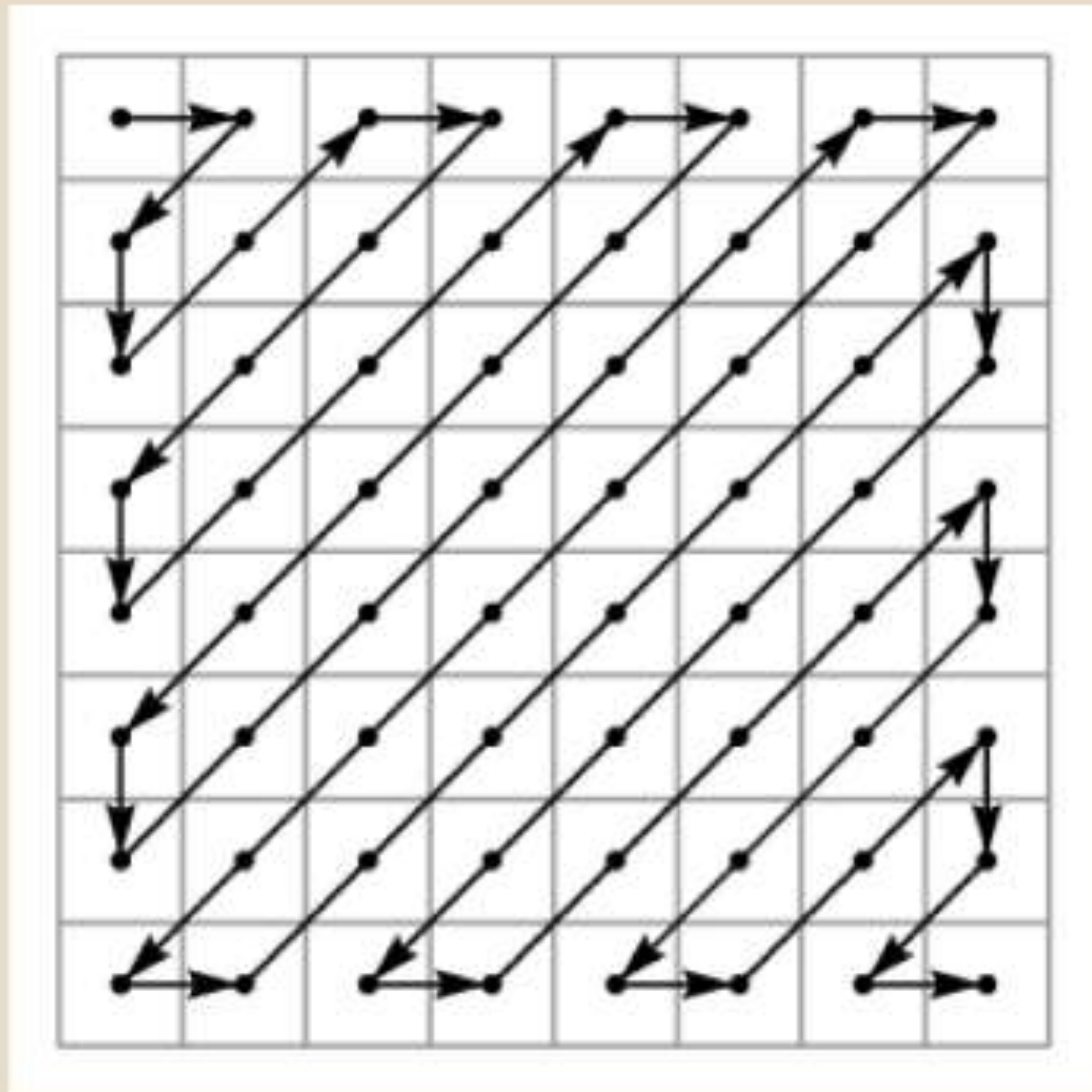


$$\begin{bmatrix}
 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\
 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\
 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\
 63 & 58 & 71 & 122 & 154 & 102 & 70 & 67 \\
 67 & 61 & 63 & 104 & 123 & 88 & 63 & 70 \\
 79 & 65 & 66 & 70 & 67 & 53 & 56 & 75 \\
 85 & 71 & 62 & 55 & 53 & 30 & 56 & 85 \\
 87 & 79 & 63 & 64 & 65 & 74 & 82 & 94 \\
 & -49 & -64 & -68 & -58 & 31 & 78 & 137 \\
 & -43 & -54 & -54 & -69 & 07 & 34 & 71 \\
 & -41 & -42 & -39 & -66 & 55 & 31 & 20 \\
 & & -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 \\
 & & -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 \\
 & & -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10
 \end{bmatrix}
 \begin{bmatrix}
 -64 & -55 \\
 -59 & -56 \\
 -62 & -55 \\
 20 & 58 \\
 27 & 59 \\
 4 & 56.12 \\
 13.15 & -20.10 \\
 -28.91 & -2.39 \\
 -10.24 & 0.46 \\
 -1.87 & -8.54 \\
 -2.38 & 4.88 \\
 0.94 & 5.42 \\
 4.30 & -5.65 \\
 1.85 & 1.95 \\
 -0.66 & 3.14 \\
 1.68 & 1.85
 \end{bmatrix}$$

## Forward Discrete Cosine Transform

Reference: [How JPG works](#)





$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

-26,-3,0,-3,-2,-6,2,-4,1,-3,1,1,5,  
1,2,-1,1,-1,2,0,0,0,0,0,-1,-1,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

## Statistical encoding

Reference: [How JPG works](#)



# Progressive JPGs



Image source: [What is a progressive JPEG?](#)



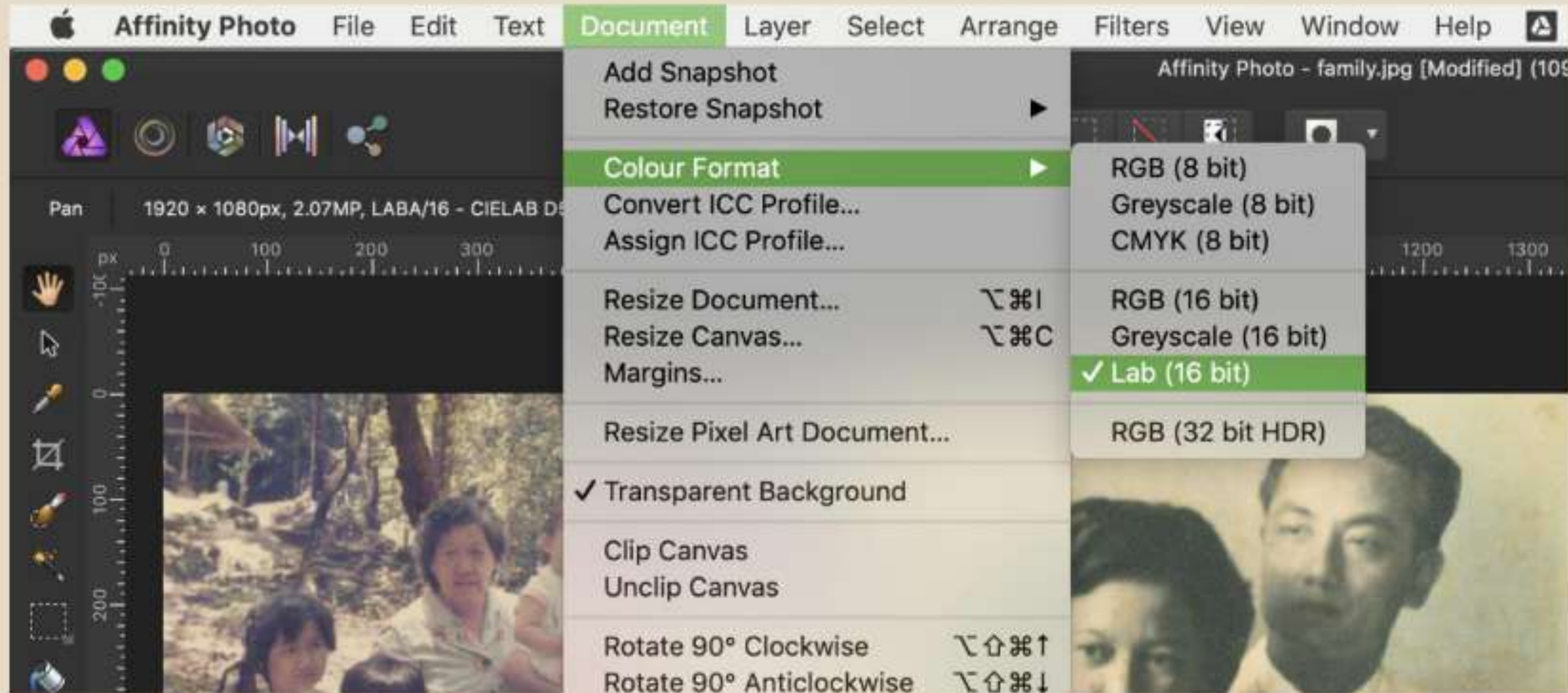
# JPG optimisation tips

1. Use high quality source material
2. Alignment on the 8x8 pixel grid
3. Reduce contrast and saturation
4. Sepia images
5. Slight blurring

Reference: [Finally understanding JPG](#)



# 3. Reduce contrast and saturation



Reference: [Finally understanding JPG](#)




## 2. Alignment on the 8x8 pixel grid



Reference: [Finally understanding JPG](#)




## 4. Sepia images



✗ FILL ✗

```
.octo-bg {  
  height: 400px;  
  background: url(img/octopus.svg);  
  background-repeat: no-repeat;  
  filter: brightness(.7) contrast(2)  
  sepia(1) hue-rotate(60deg) saturate(2);  
}
```



< >

Una Kravets: CSS Blend Modes, Because...

Reference: [Finally understanding JPG](#)



# 5. Slight blurring



opt-blurred.jpg

JPEG image - 59 KB

**59 KB**

Tags Add Tags...  
Created Today, 18:10  
Modified Today, 18:10  
Content created Tuesday, 23 April 2019 at 18:10  
Dimensions 800x800  
Colour space RGB  
[Show Less](#)



opt-notblurred.jpg

JPEG image - 65 KB

**65 KB**

Tags Add Tags...  
Created Today, 18:10  
Modified Today, 18:10  
Content created Tuesday, 23 April 2019 at 18:10  
Dimensions 800x800  
Colour space RGB

Reference: [Finally understanding JPG](#)



# Speed, Quality, Size



Can't have 'em all



# JPG encoders, there are many



libjpeg



mozJPEG



Guetzli



# libjpeg-turbo

libjpeg-turbo Home

About libjpeg-turbo

Professional Services

Sponsors

Software That Uses or Provides libjpeg-turbo

SIMD Coverage of the libjpeg Algorithms

"libjpeg-turbo" is "TurboJPEG"

Mailing Lists

Downloads

Digital Signatures

Official Binaries

Supported Platforms and Other Notes

VHM Repository

Documentation

Reports

libjpeg-turbo Performance Study

A Study on the Usefulness of DCT Scaling and SmartScale

Other Reports

Position Statements

RJD

What About libjpeg v9?

What About mjpeg?

Developer Info

Git Access

Pre-Release Builds/Continuous Integration

Build Instructions

Contact

libjpeg-turbo

libjpeg-turbo is a JPEG image codec that uses SIMD instructions (MMX, SSE2, AVX2, NEON, AltiVec) to accelerate baseline JPEG compression and decompression on x86, x86-64, ARM, and PowerPC systems, as well as progressive JPEG compression on x86 and x86-64 systems. On such systems, libjpeg-turbo is generally 2-6x as fast as libjpeg, all else being equal. On other types of systems, libjpeg-turbo can still outperform libjpeg by a significant amount, by virtue of its highly-optimized Huffman coding routines. In many cases, the performance of libjpeg-turbo rivals that of proprietary high-speed JPEG codecs.

libjpeg-turbo implements both the traditional libjpeg API as well as the less powerful but more straightforward TurboJPEG API. libjpeg-turbo also features colorspace extensions that allow it to compress/decode/convert to 32-bit and big-endian pixel buffers (JRGB, XRGB, etc.), as well as a full-featured Java interface.

libjpeg-turbo was originally based on [libjpeg-SIMD](#), an MMX-accelerated derivative of libjpeg v6b developed by Miyasaka Masaru. The TigerVNC and VirtualGL projects made numerous enhancements to the codec in 2008, and in early 2010, libjpeg-turbo spun off into an independent project, with the goal of making high-speed JPEG compression/decompression technology available to a broader range of users and developers.

GitHub [Project Page](#) (Code repository, issues/feature trackers)

[Mailing Lists](#) (Google Groups)

[Official Binaries and Source Tarballs](#) 

Donate



[Donate using Bitcoin/Ethereum/Litecoin](#)

If you have benefited from this product, then please consider making a donation to ensure that we can continue to provide this enterprise-quality, high-performance software free of charge and in a vendor-neutral manner. Every dollar donated goes toward the development of libjpeg-turbo. Alternatively, if you have any improvements in mind for the product, please consider [funding the labor](#) necessary to implement them.

[\\*\\*\\* Projects in need of funding \\*\\*\\*](#)

NEWS

2019-02-04: [libjpeg-turbo Becomes Official ISO/ITU-T Reference Implementation](#)

2019-01-01: [A New Year and a Desperate Plea for Help](#)

2016-05-02: [libjpeg-turbo Named SourceForge Project of the Month, May 2016](#)

 All content on this web site is licensed under the [Creative Commons Attribution 4.0 International License](#). Any work containing material derived from this web site must carry the Creative Commons Project Attribution license of the material and/or the license URL for this license (CC BY 4.0).

[Link](#) - [History](#) - [Print](#) - [Recent Changes](#) - [Search](#)

Page last modified on February 02, 2019, at 09:54 AM

<https://libjpeg-turbo.org/>



# GIF89a





# Burn All GIFs day

November 5, 1999



<https://burnallgifs.org/archives/>



# PNG filter algorithms

None	—	Zero	$\begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & 59 & 55 & 90 & 109 \end{array} \longrightarrow \begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & 59 & 55 & 90 & 109 \end{array}$
Sub	$\text{Sub}(x) = \text{Raw}(x) - \text{Raw}(x-\text{bpp})$	Byte A (to the left)	$\begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & 59 & 55 & 90 & 109 \end{array} \longrightarrow \begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & -4 & -4 & 35 & 19 \end{array}$
Up	$\text{Up}(x) = \text{Raw}(x) - \text{Prior}(x)$	Byte B (above)	$\begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & 59 & 55 & 90 & 109 \end{array} \longrightarrow \begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & 4 & -6 & 24 & 39 \end{array}$
Average	$\text{Average}(x) = \text{Raw}(x) - \text{floor}((\text{Raw}(x-\text{bpp}) + \text{Prior}(x))/2)$	Mean of bytes A and B, rounded down	$\begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & 59 & 55 & 90 & 109 \end{array} \longrightarrow \begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & 0 & -5 & 30 & 29 \end{array}$
Paeth	$\text{Paeth}(x) = \text{Raw}(x) - \text{PaethPredictor}(\text{Raw}(x-\text{bpp}), \text{Prior}(x), \text{Prior}(x-\text{bpp}))$	A, B, or C, whichever is closest to $p = A + B - C$	$\begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & 59 & 55 & 90 & 109 \end{array} \longrightarrow \begin{array}{ccccc} 52 & 55 & 61 & 66 & 70 \\ 63 & -4 & -4 & 35 & 19 \end{array}$



# Portable Network Graphics (PNG)

The image shows a hex editor window with the following components:

- Hex View:** Displays the raw bytes of the file. The first 170 bytes are shown, with the 8-byte signature highlighted in green and labeled **89 50 4e 47 0d 0a 1a 0a**.
- ASCII View:** Displays the corresponding ASCII characters for the hex data. The signature bytes correspond to the characters `89504e470d0a1a0a`.
- Data Type Interpretation:** A panel on the right shows the data offset (0x0=0) and various data types interpreted from the hex data, including `hex8` (0x89), `int8` (-119), `int16` (-65399), `int32` (-4294967159), `int64` (-18446744073706), `uint8` (137), `uint16` (137), `uint32` (137), `uint64` (137), `float`, and `double`.

8-byte signature of a PNG file



# DEFLATE compression algorithm

[\[Docs\]](#) [\[txt|pdf\]](#) [\[draft-deutsch-d...\]](#) [\[Tracker\]](#) [\[Diff1\]](#) [\[Diff2\]](#)

INFORMATIONAL

Network Working Group  
Request for Comments: 1951  
Category: Informational

P. Deutsch  
Aladdin Enterprises  
May 1996

## **DEFLATE Compressed Data Format Specification version 1.3**

### Status of This Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

### IESG Note:

The IESG takes no position on the validity of any Intellectual Property Rights statements contained in this document.

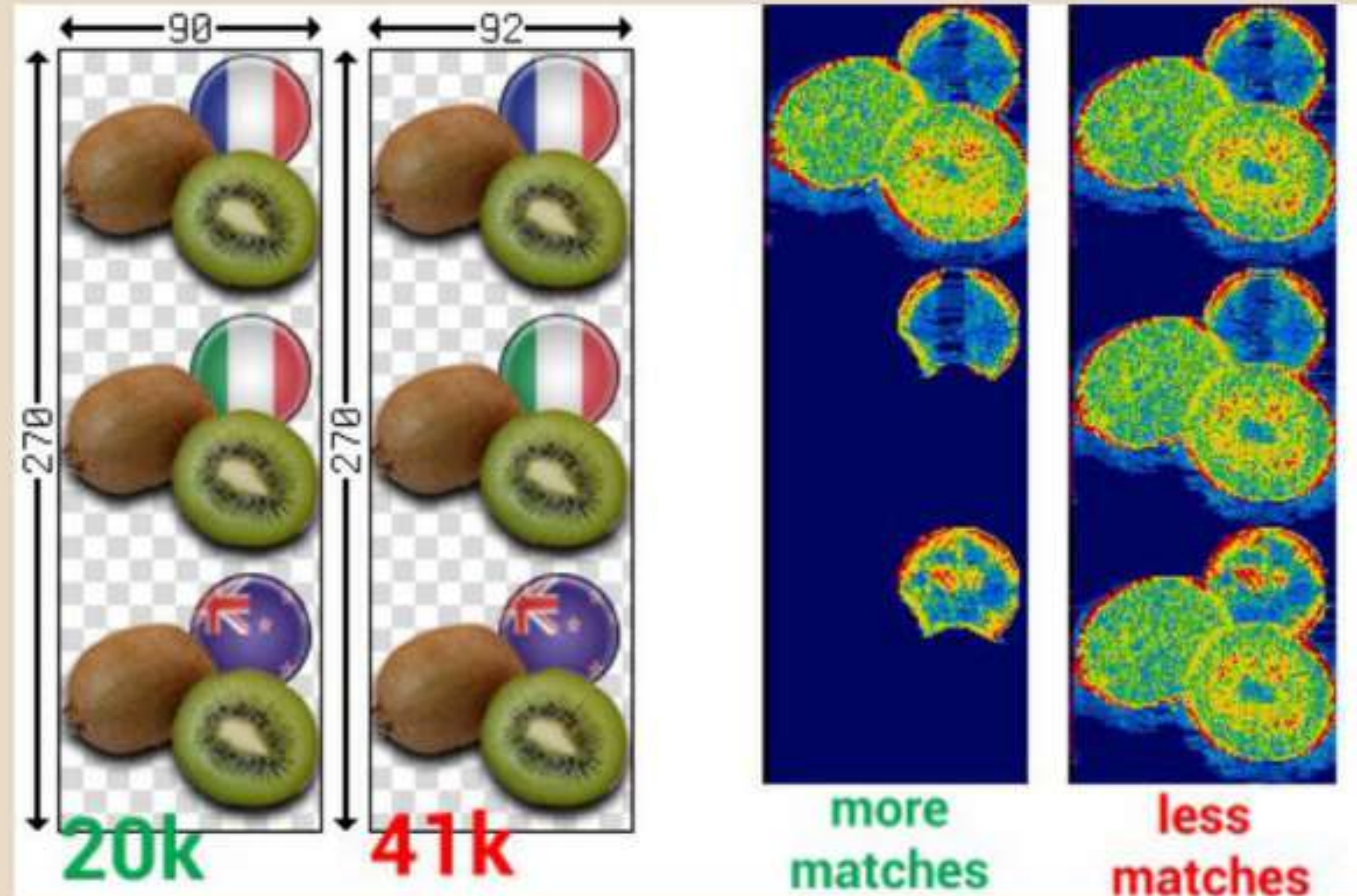
### Notices

Copyright (c) 1996 L. Peter Deutsch

**DEFLATE Compressed Data Format Specification version 1.3**



# What a difference 2 pixels can make



Reference: [pngthermal](#)



# Optimising PNG files

1. Reduce number of colours
2. Choose the right pixel format
3. Use indexed images, if possible
4. Optimise fully transparent pixels

Reference: [Reducing PNG file Size](#)



### 3. Use indexed images, if possible



Reference: [Reducing PNG file Size](#)



## 4. Optimise fully transparent pixels



Masked portion of image filled with single colour

Masked portion of image untouched

Reference: [Reducing PNG file Size](#)



# libpng

## libpng

**libpng** is the official PNG reference library. It supports almost all PNG features, is extensible, and has been extensively tested for over 23 years. The home site for development versions (i.e., may be buggy or subject to change or include experimental features) is <https://libpng.sourceforge.io/>, and the place to go for questions about the library is the [png-mng-implement](#) mailing list.

libpng is available as ANSI C (C89) source code and requires **zlib 1.0.4** or later (**1.2.5** or later recommended for performance and security reasons). The current public release, **libpng 1.6.36**, fixes some build issues, adds a couple of small optimizations (ARM `png_do_expand_palette()`, Intel SSE2 `memcpy()`), and updates the license (identical terms to the zlib license, with the old license appended in the manner of the Python Software Foundation License version 2, and the list of contributing authors moved to a separate AUTHORS file).

### Portability Note

The libpng 1.5.x, 1.6.x, and upcoming 1.7.x series continue the evolution of the libpng API, finally **hiding the contents** of the venerable and hoary `png_struct` and `png_info` data structures inside **private (i.e., non-installed) header files**. Instead of direct struct-access, applications should be using the various `png_get_xxx()` and `png_set_xxx()` accessor functions, which have existed for almost as long as libpng itself.

The portability notice should not come as a particular surprise to anyone who has added libpng support to an application this millenium; the manual has warned of it since at least July 2000. (Specifically: *"Starting with version 2.0.0, both structures are going to be hidden, and the contents of the structures will only be accessible through the `png_get/png_set` functions."* OK, so the version number was off a bit...and the grammar, too, but who's counting?) Those whose apps depend on the older API need not panic, however (for now); libpng 1.2.x continues to get security fixes, as has 1.0.x for well over a decade. (Greg no longer bothers to list either series here; enough's enough, folks. Update those apps now!)

The 1.5.x and later series also include a new, more thorough test program (`pngvalid.c`) and a new `pnglibconf.h` header file that tracks what features were enabled or disabled when libpng was built. On the other hand, they no longer internally include the `zlib.h` header file, so applications that formerly depended on `png.h` to provide that will now need to include it explicitly. Complete differences relative to libpng 1.4.x are detailed [here](#).

See the bottom of this page for warnings about **security and crash bugs** in versions up through **libpng 1.6.31**.

In addition to the main library sources, all of the 1.2.x/1.4.x/1.5.x/1.6.x/1.7.x series include the [rpng](#), [rpng2](#) and [wpng](#) demo programs, the `pngminus` demo program, a subset of Willem van Schaik's [PngSuite test images](#), and Willem's VisualPng demo program.

<http://www.libpng.org/pub/png/libpng.html>



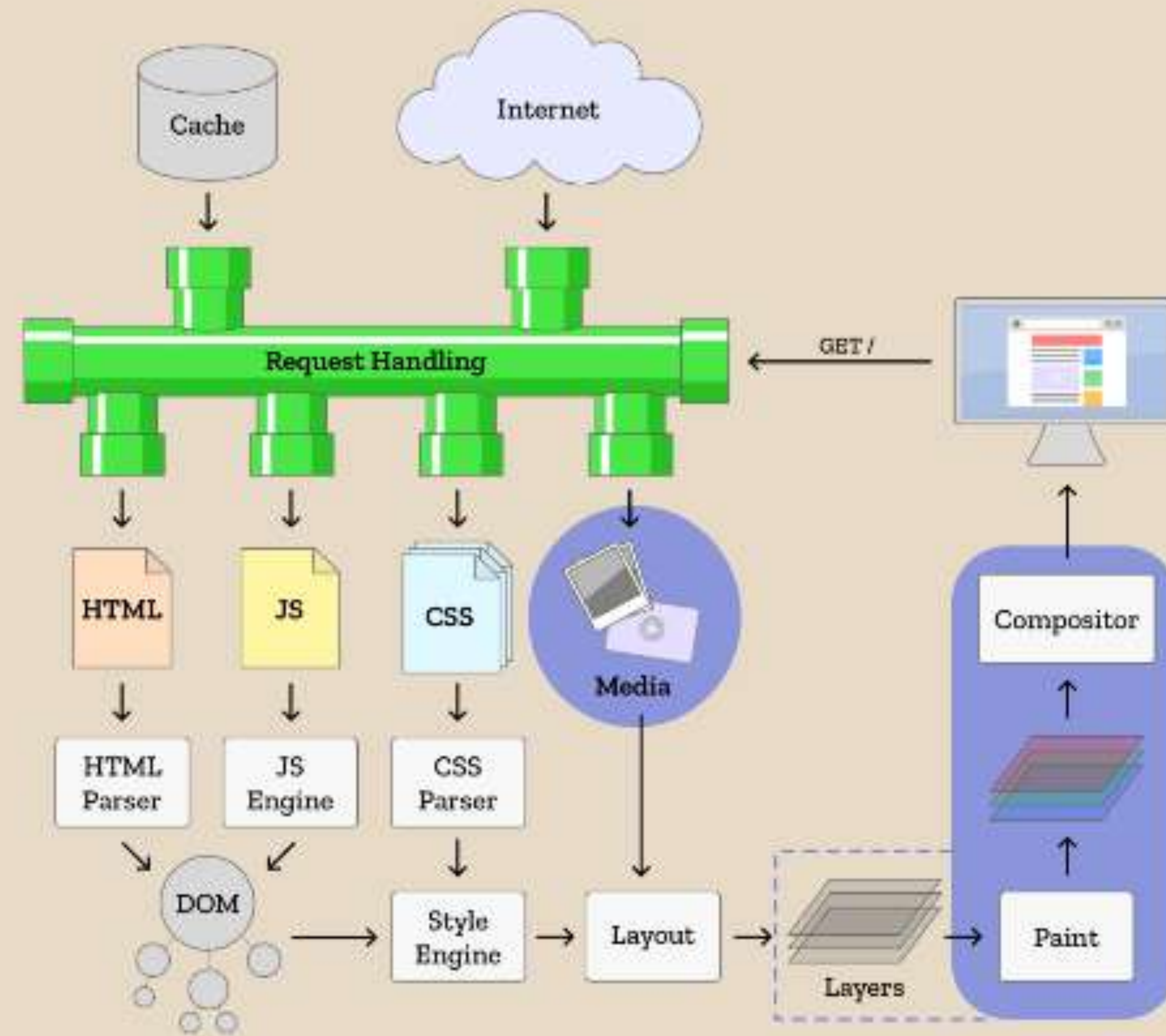
# What is a browser engine? 🤔



Source: [Quantum Up Close: What is a browser engine?](#)



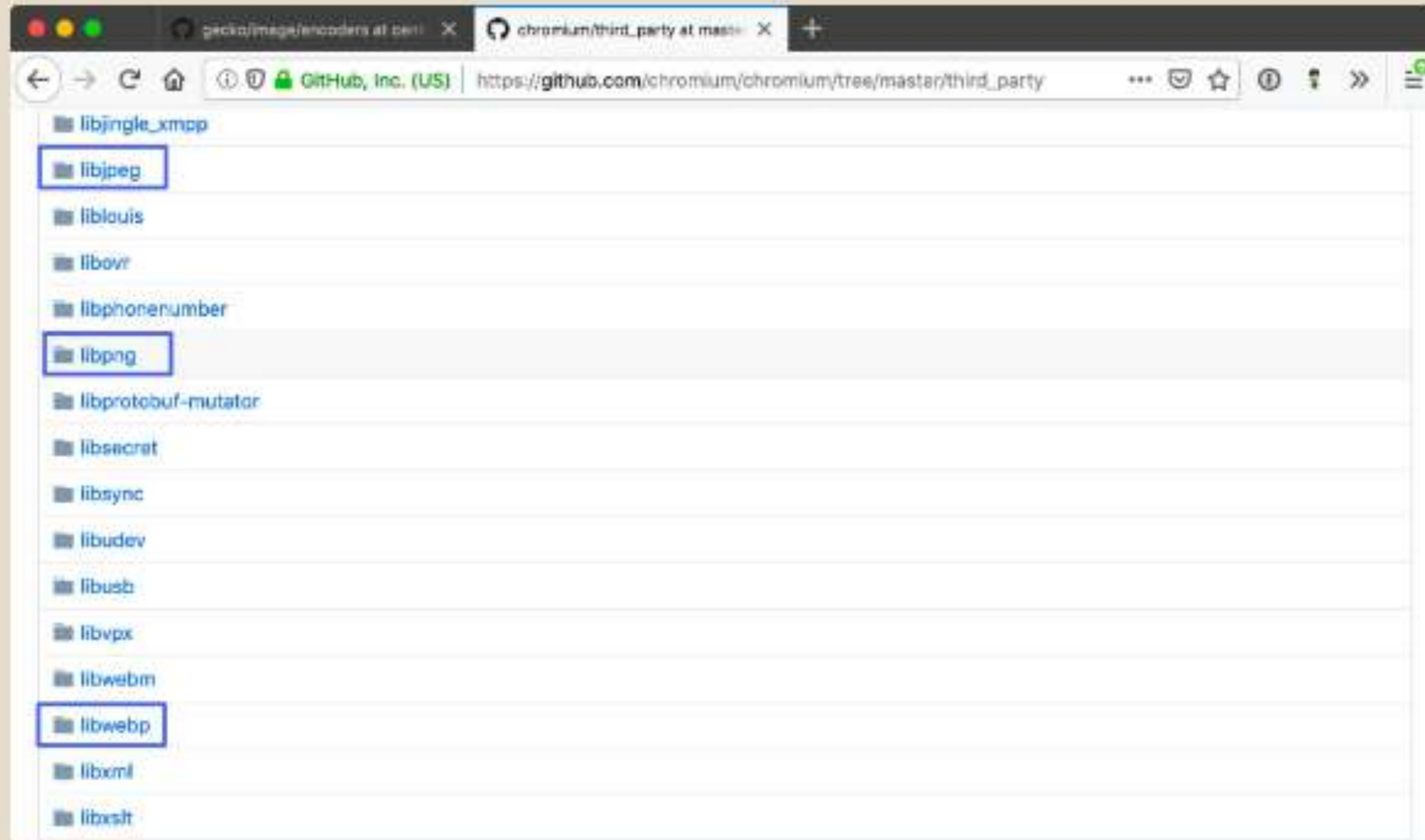
# What is a browser engine? 🧐



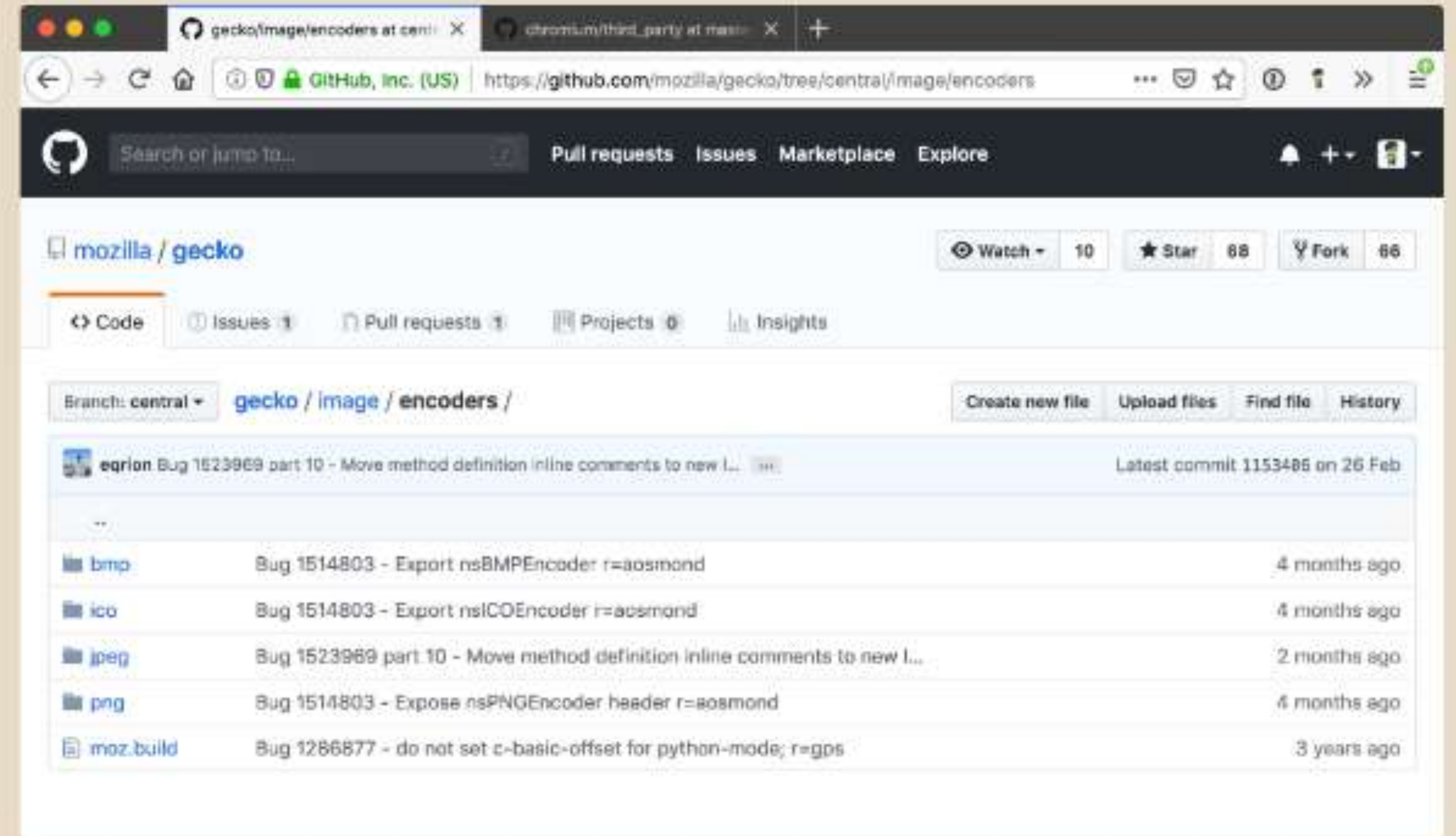
Source: [Quantum Up Close: What is a browser engine?](#)



# Image encoders in browsers



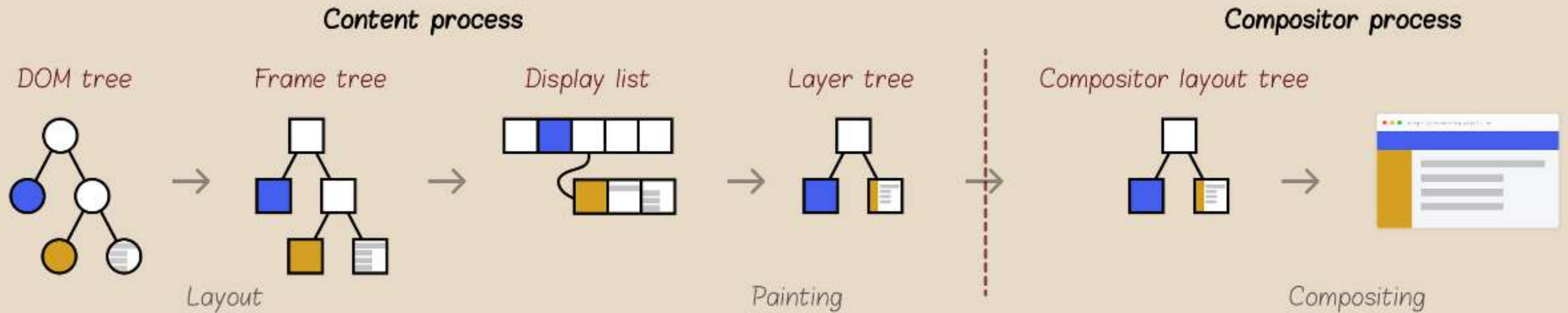
Chromium



Gecko

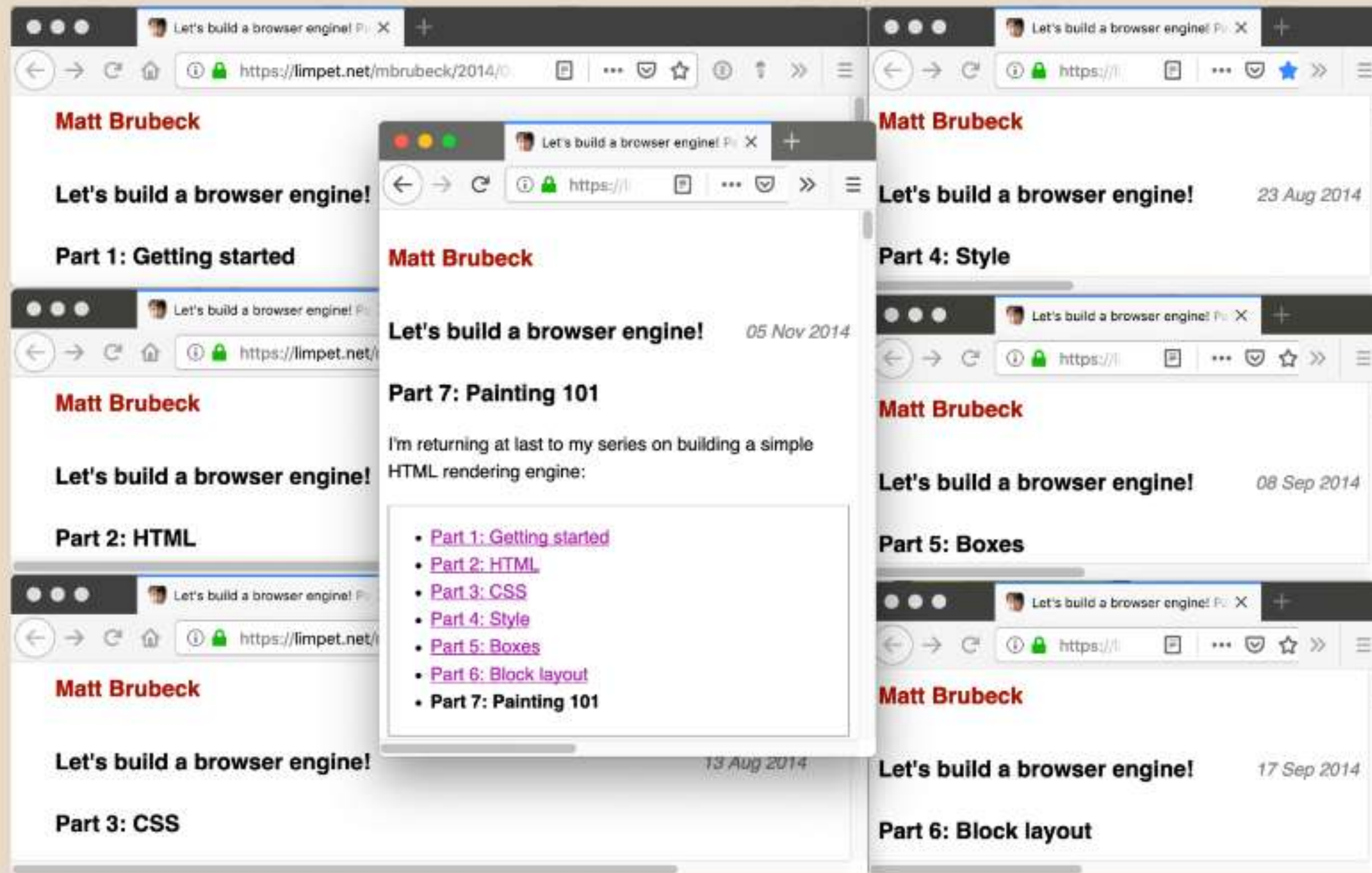


# Browser rendering pipeline



Reference: [Introduction to WebRender – Part 1 – Browsers today](#)





Reference: [Let's build a browser engine!](#)



# Rasterisation (1/3)

Simple rasteriser for painting rectangles

```
pub struct Canvas {  
    pub pixels: Vec<Color>,  
    pub width: usize,  
    pub height: usize,  
}  
  
impl Canvas {  
    /// Create a blank canvas  
    fn new(width: usize, height: usize) → Canvas {  
        let white = Color { r: 255, g: 255, b: 255, a: 255 };  
        Canvas {  
            pixels: vec![white; width * height],  
            width: width,  
            height: height,  
        }  
    }  
    // ...  
}
```

Source: [Let's build a browser engine!](#)



# Rasterisation (2/3)

Simple rasteriser for painting rectangles

```
fn paint_item(&mut self, item: &DisplayCommand) {  
    match *item {  
        DisplayCommand::SolidColor(color, rect) => {  
            // Clip the rectangle to the canvas boundaries.  
            let x0 = rect.x.clamp(0.0, self.width as f32) as usize;  
            let y0 = rect.y.clamp(0.0, self.height as f32) as usize;  
            let x1 = (rect.x + rect.width).clamp(0.0, self.width as f32) as usize;  
            let y1 = (rect.y + rect.height).clamp(0.0, self.height as f32) as usize;  
  
            for y in y0 .. y1 {  
                for x in x0 .. x1 {  
                    self.pixels[y * self.width + x] = color;  
                }  
            }  
        }  
    }  
}
```

Source: [Let's build a browser engine!](#)



# Rasterisation (3/3)

Simple rasteriser for painting rectangles

```
/// Paint a tree of LayoutBoxes to an array of pixels.
pub fn paint(layout_root: &LayoutBox, bounds: Rect) → Canvas {
    let display_list = build_display_list(layout_root);
    let mut canvas = Canvas::new(bounds.width as usize, bounds.height as usize);
    for item in display_list {
        canvas.paint_item(&item);
    }
    canvas
}
```

Source: [Let's build a browser engine!](#)



*“The graphics backend is an array of bytes and a for loop that goes and says: go from the left edge of the rectangle to the right edge and write the same colour over and over again into this array.”*

*—Matt Brubeck, Bay Area Rust Meetup (Nov 2014)*

¯\\_(\ツ)\\_/¯



# Graphics libraries



Skia



WebRender



Core Graphics



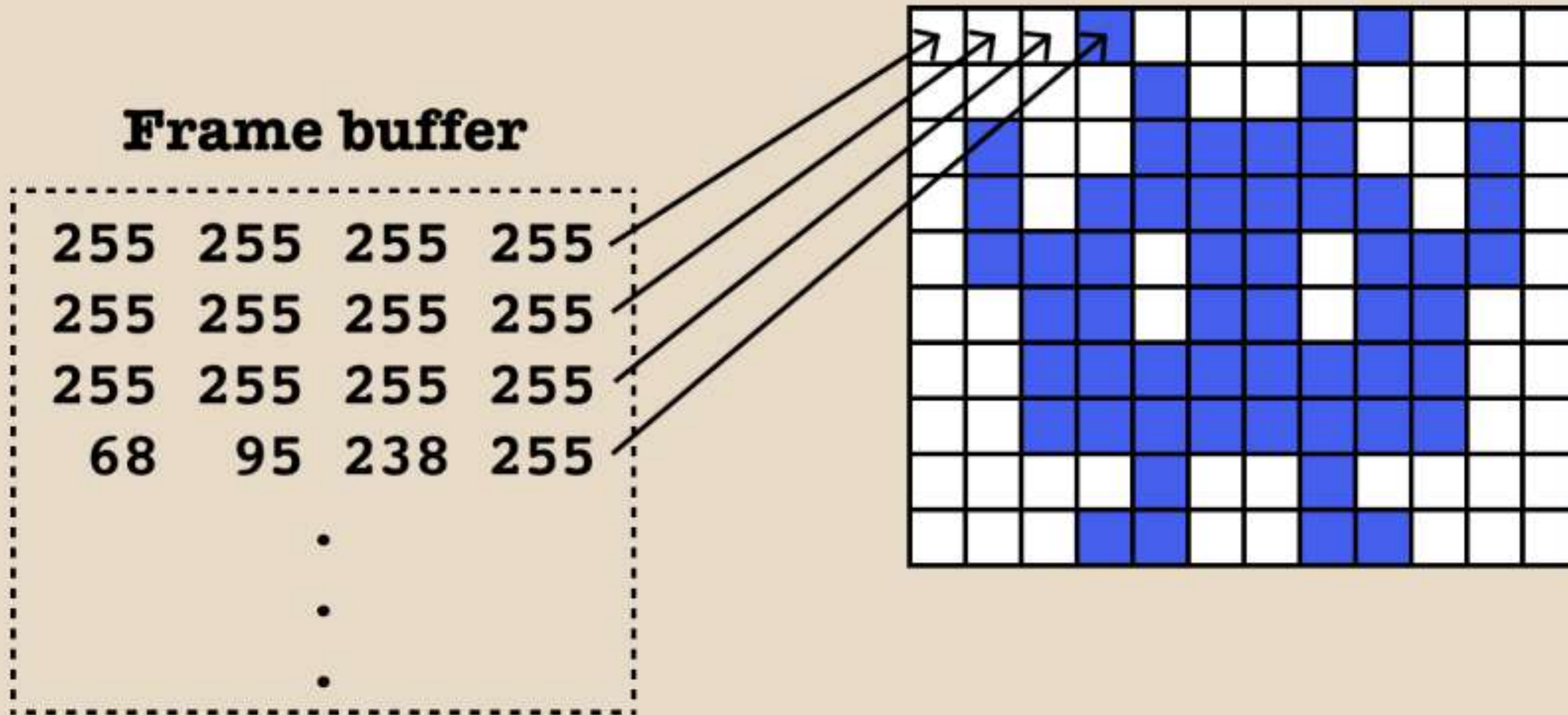
Cairo



Pango



# Painting



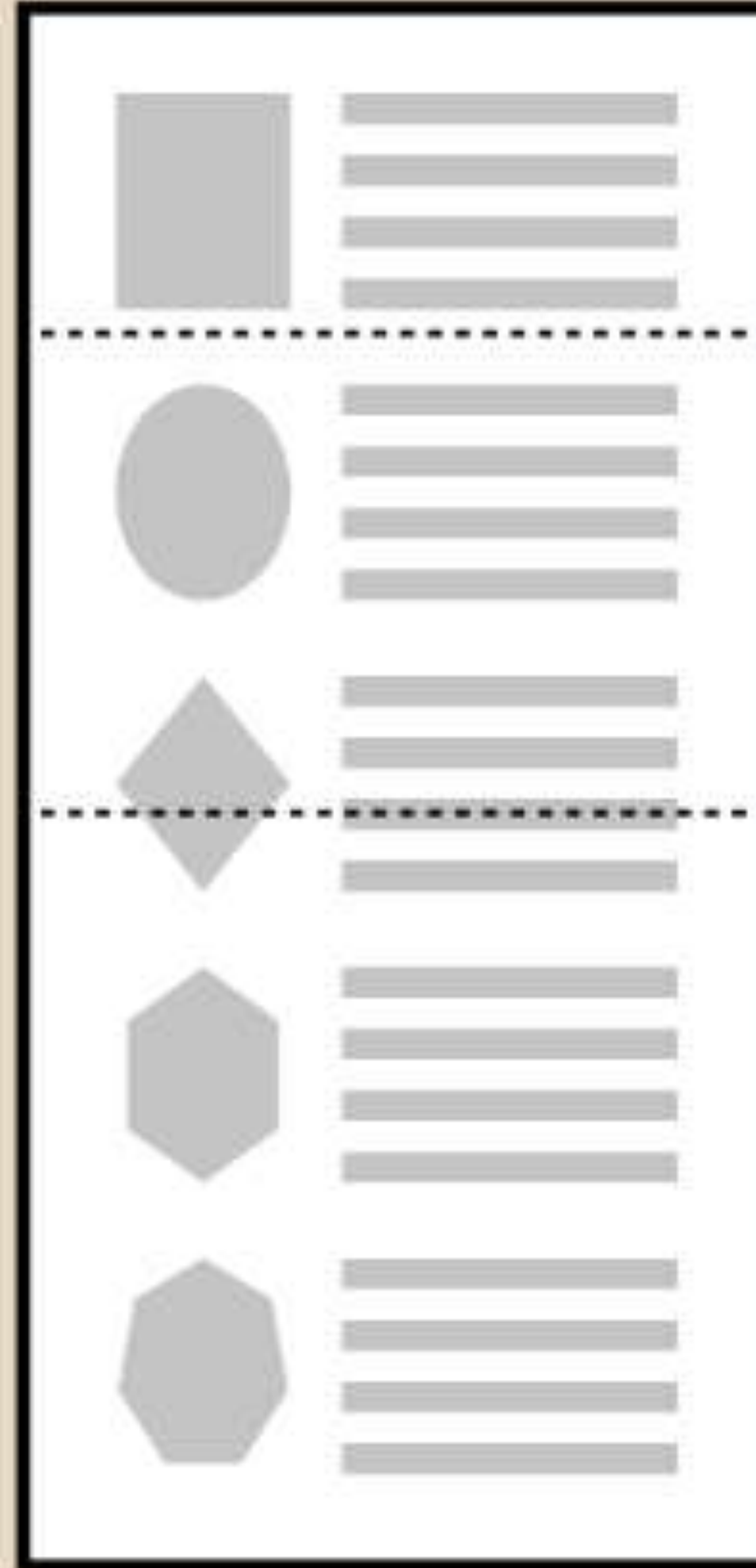
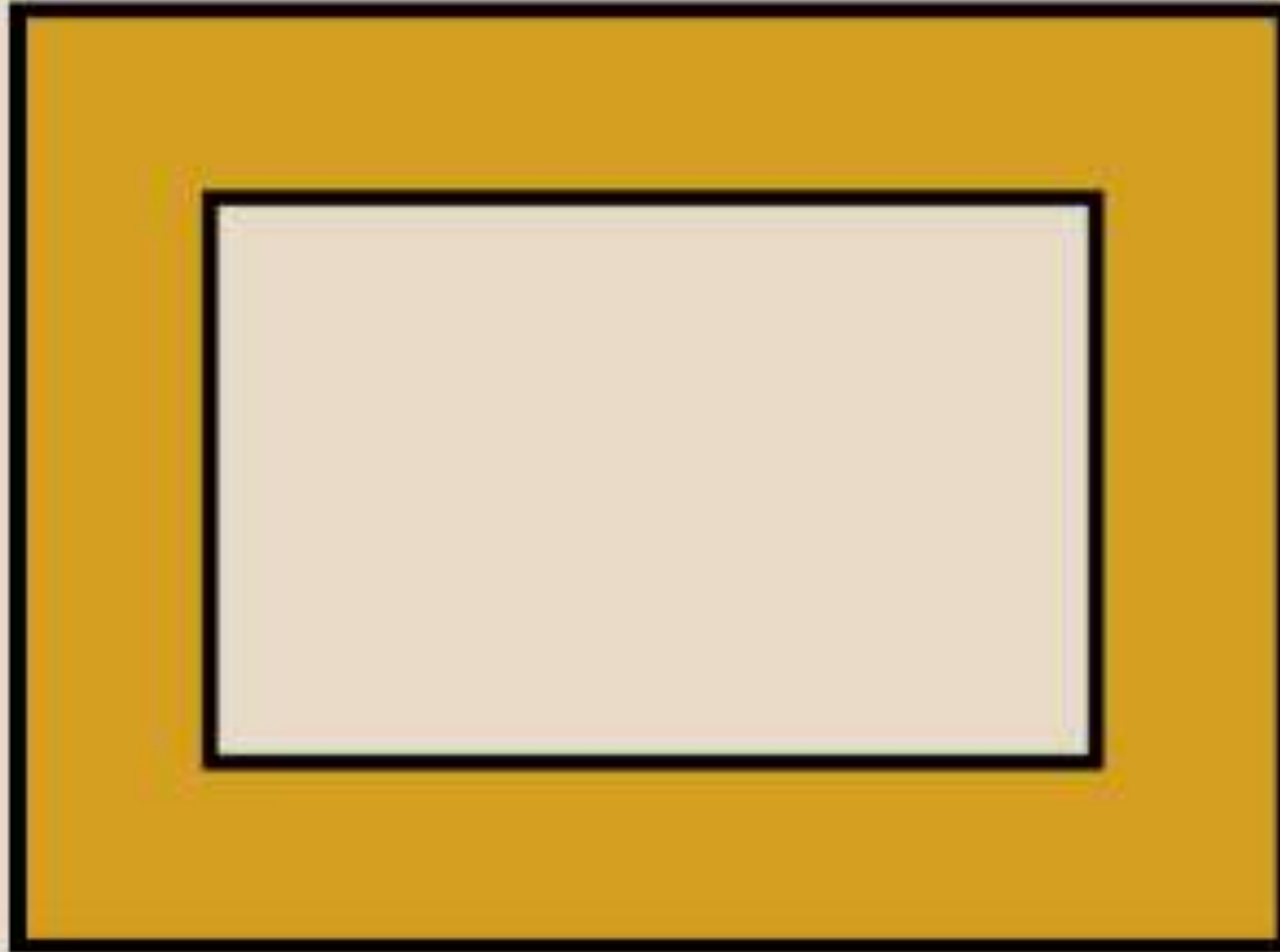
Reference: [The whole web at maximum FPS: How WebRender gets rid of jank](#)



# Compositing (1/2)

Source bitmaps

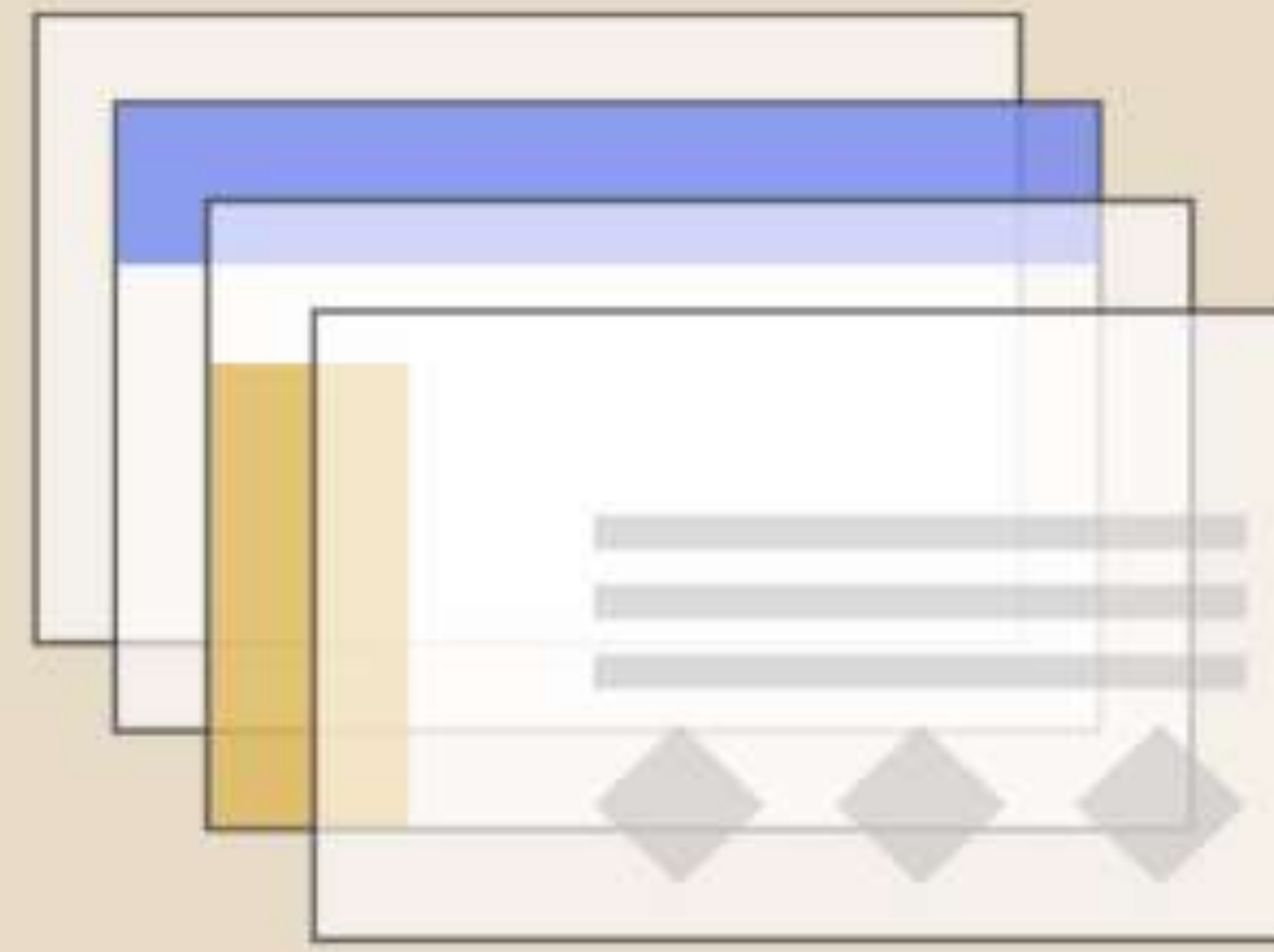
Destination bitmap



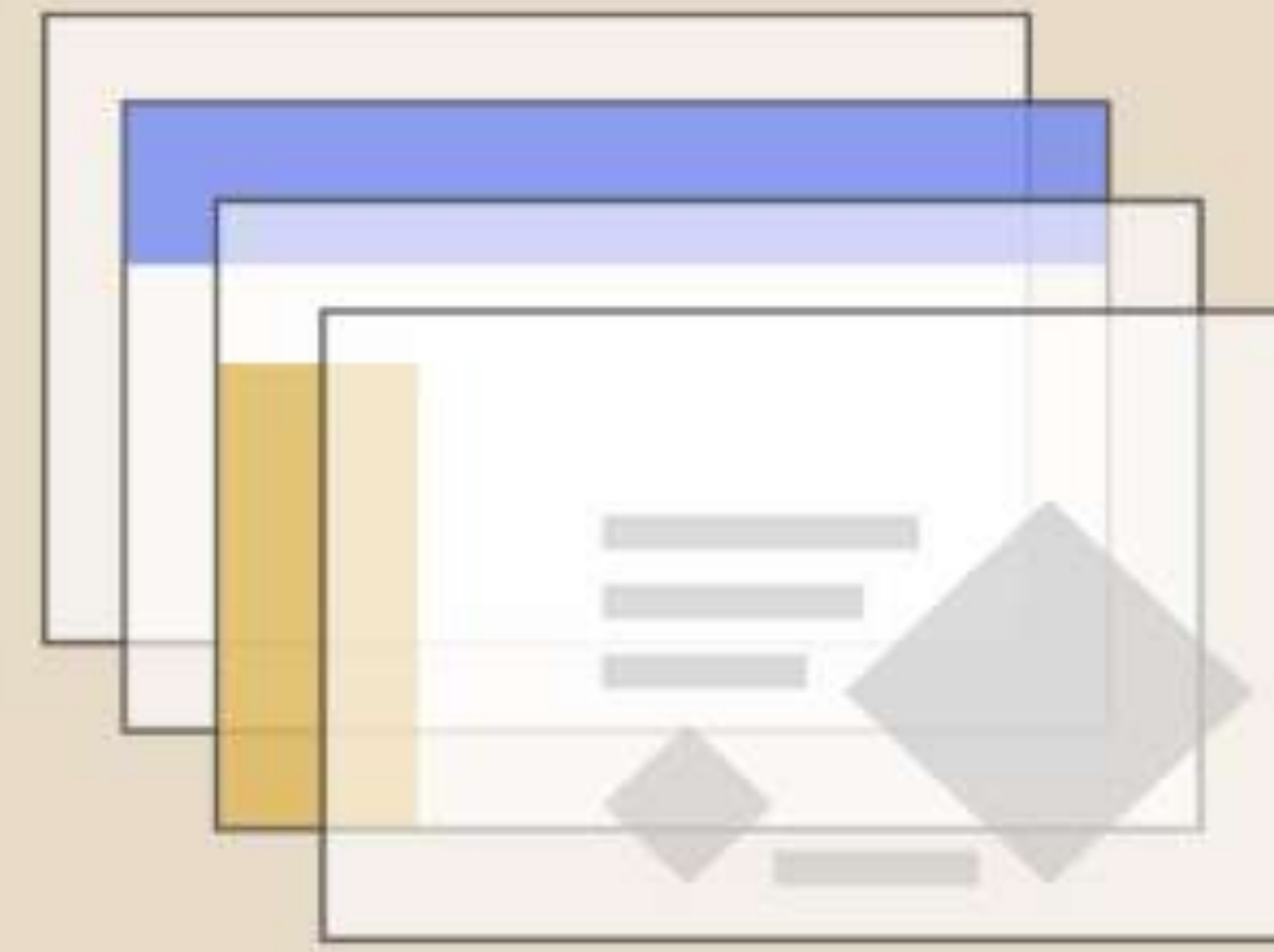
Reference: [The whole web at maximum FPS: How WebRender gets rid of jank](#)



# Compositing (2/2)



*Invalidate, repaint  
and recomposite*



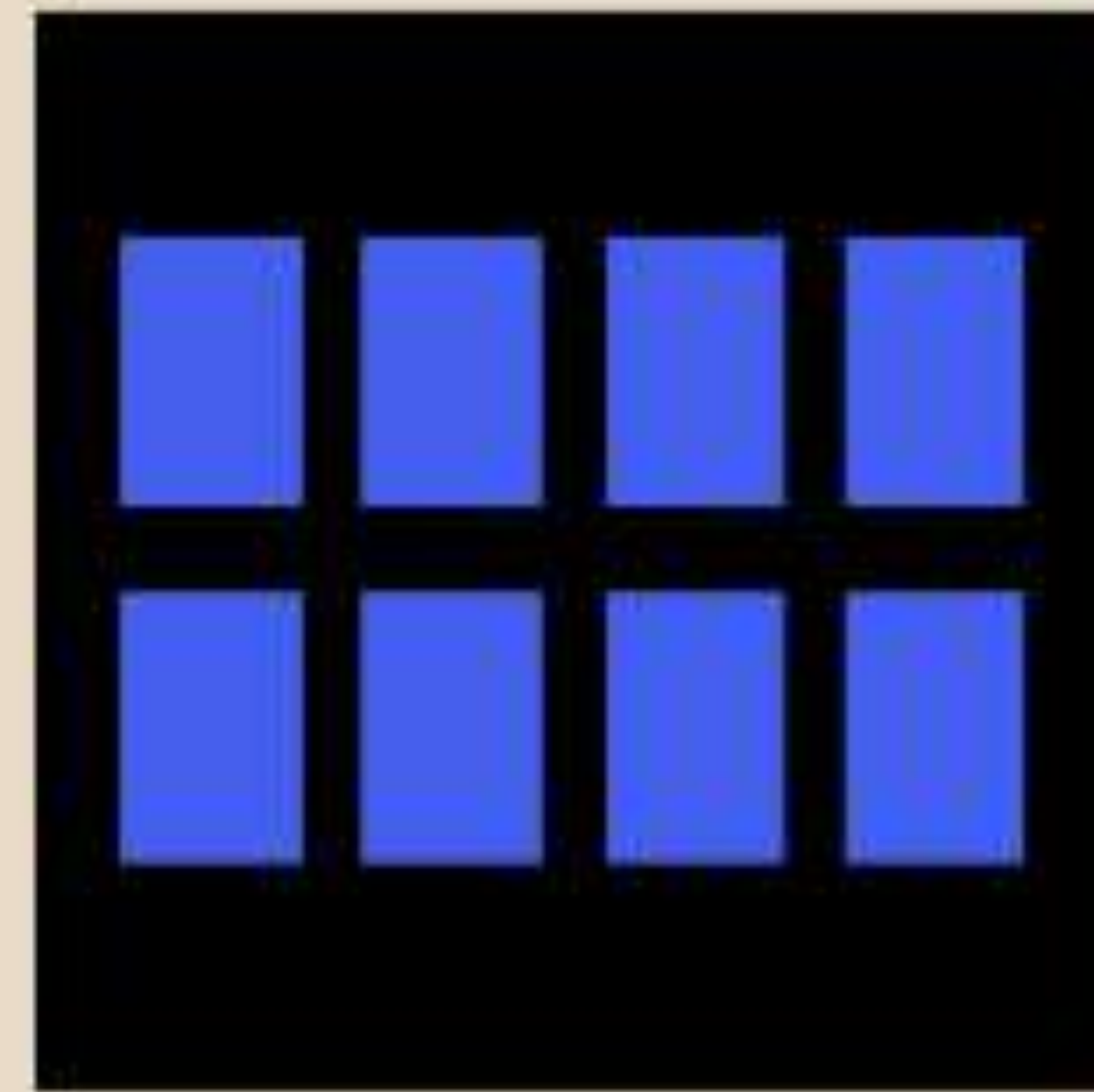
*Invalidate, repaint  
and recomposite*



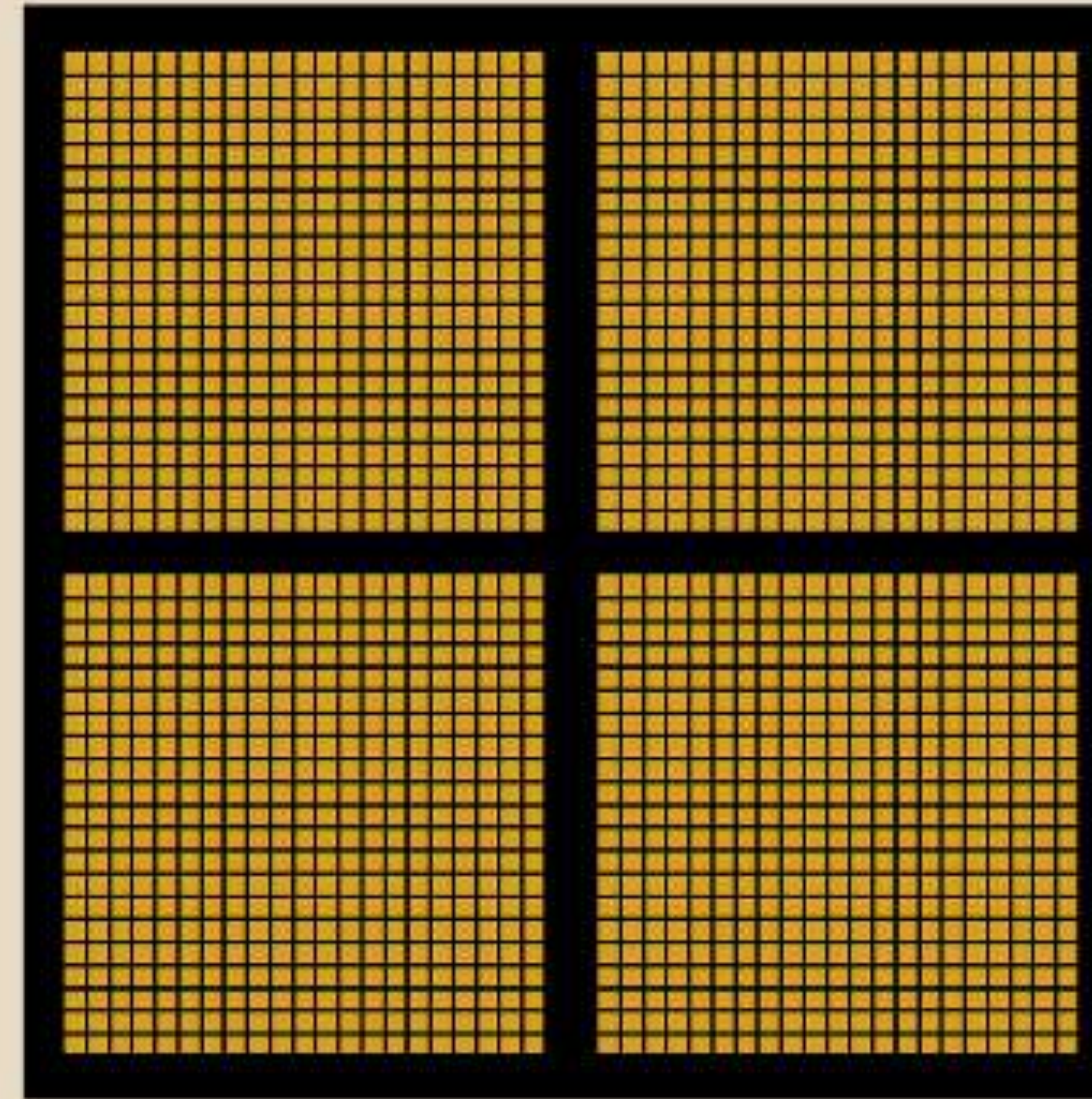
Reference: [GPU Accelerated Compositing in Chrome](#)



# Making use of the GPU (1/2)



**CPU**  
(multiple cores)



**GPU**  
(thousands of cores)



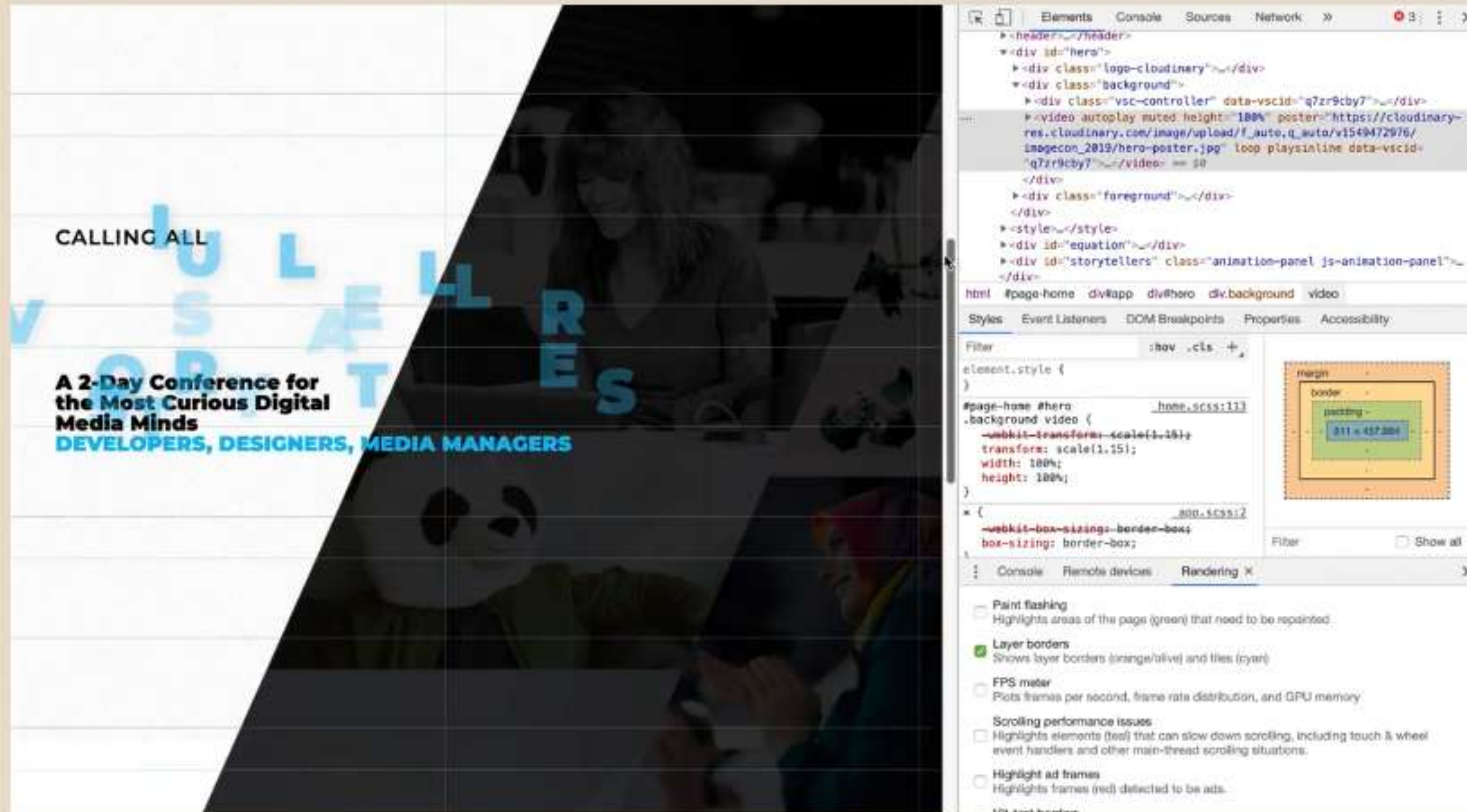
# Making use of the GPU (2/2)



Reference: [Hardware acceleration and compositing](#)



# GPU rasterisation in Chromium



Reference: [Software vs. GPU rasterization in Chromium](#)



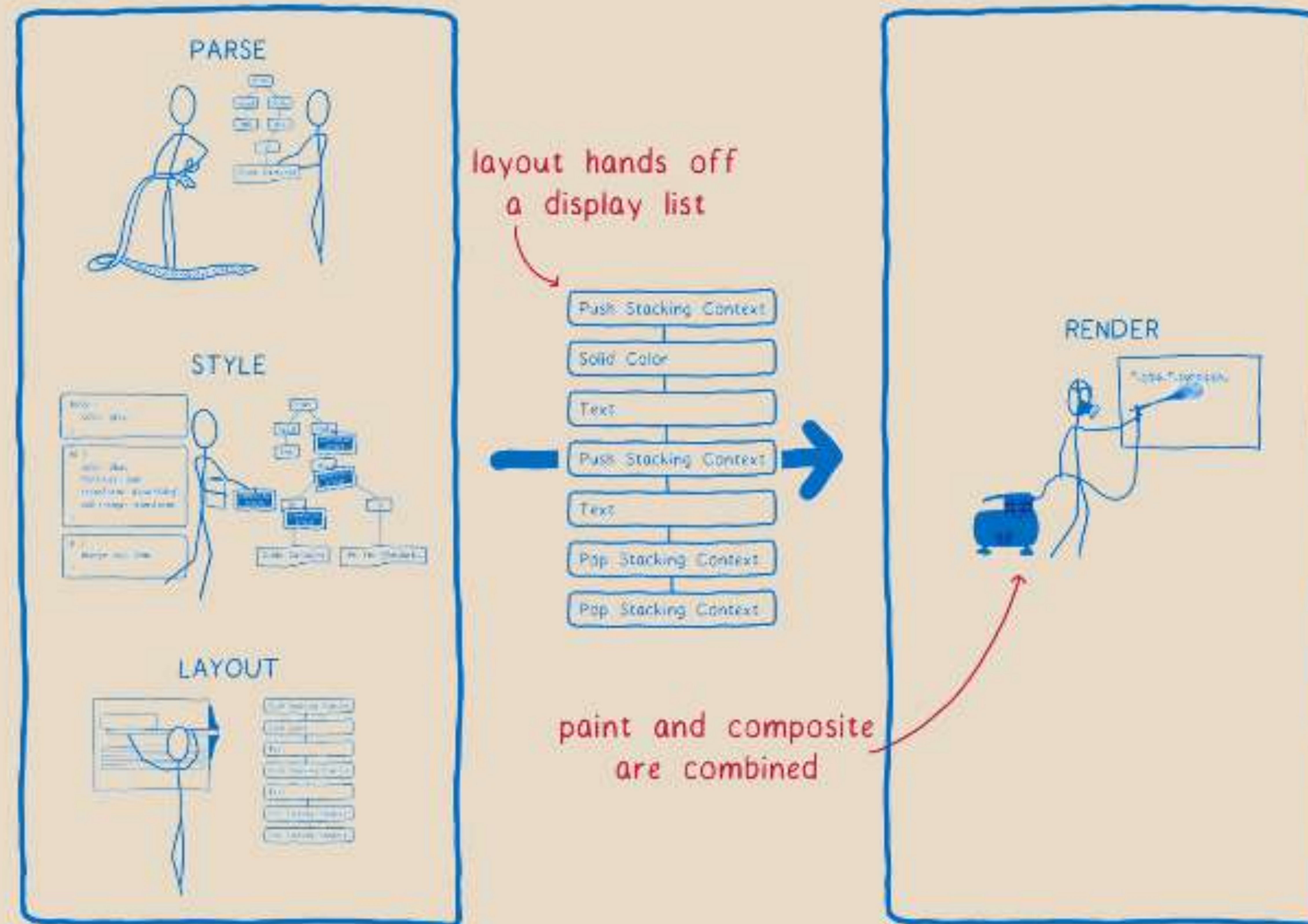
# If you could have a do-over...



What if we actually need a butterfly instead?



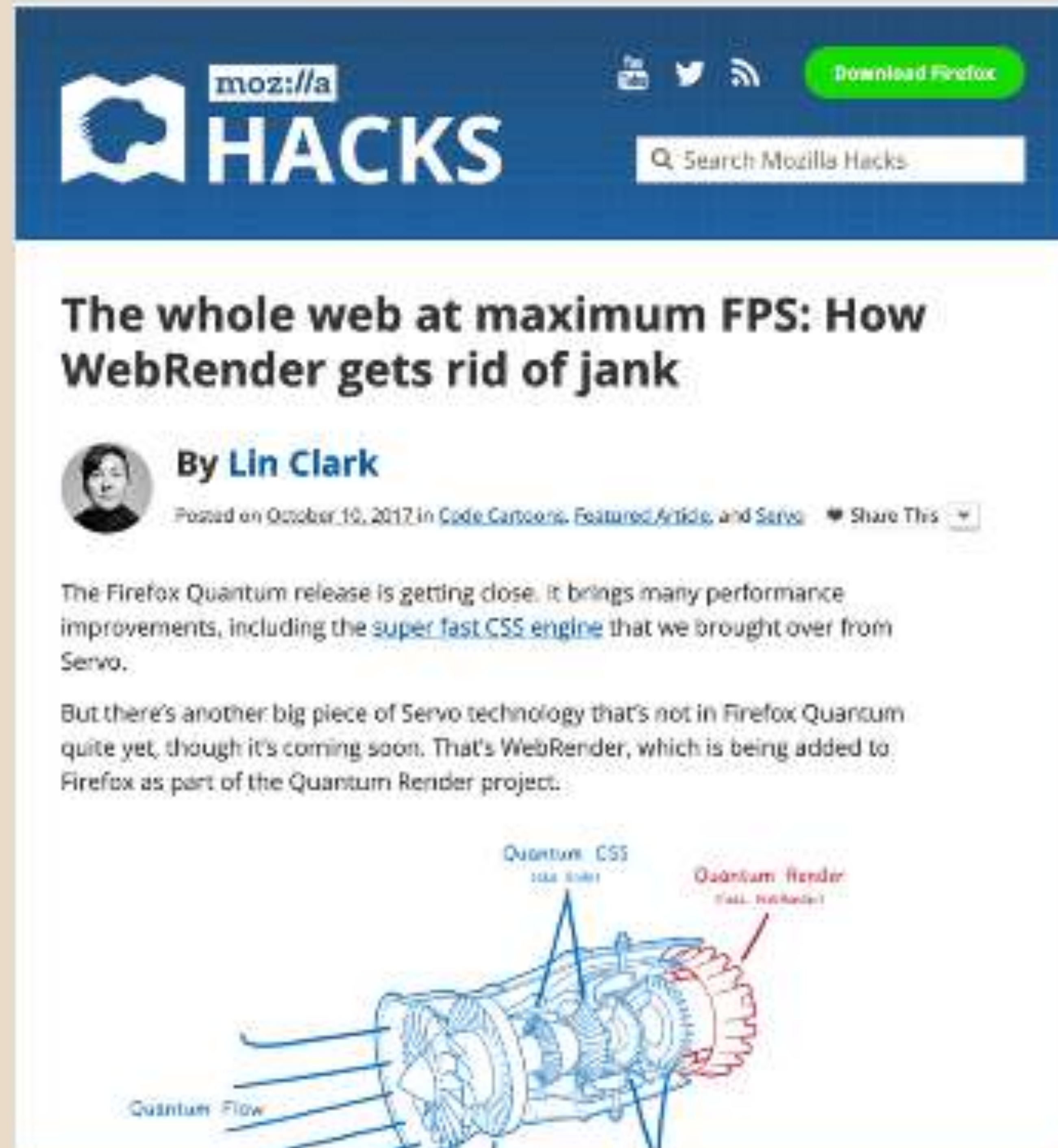
# WebRender (1/2)



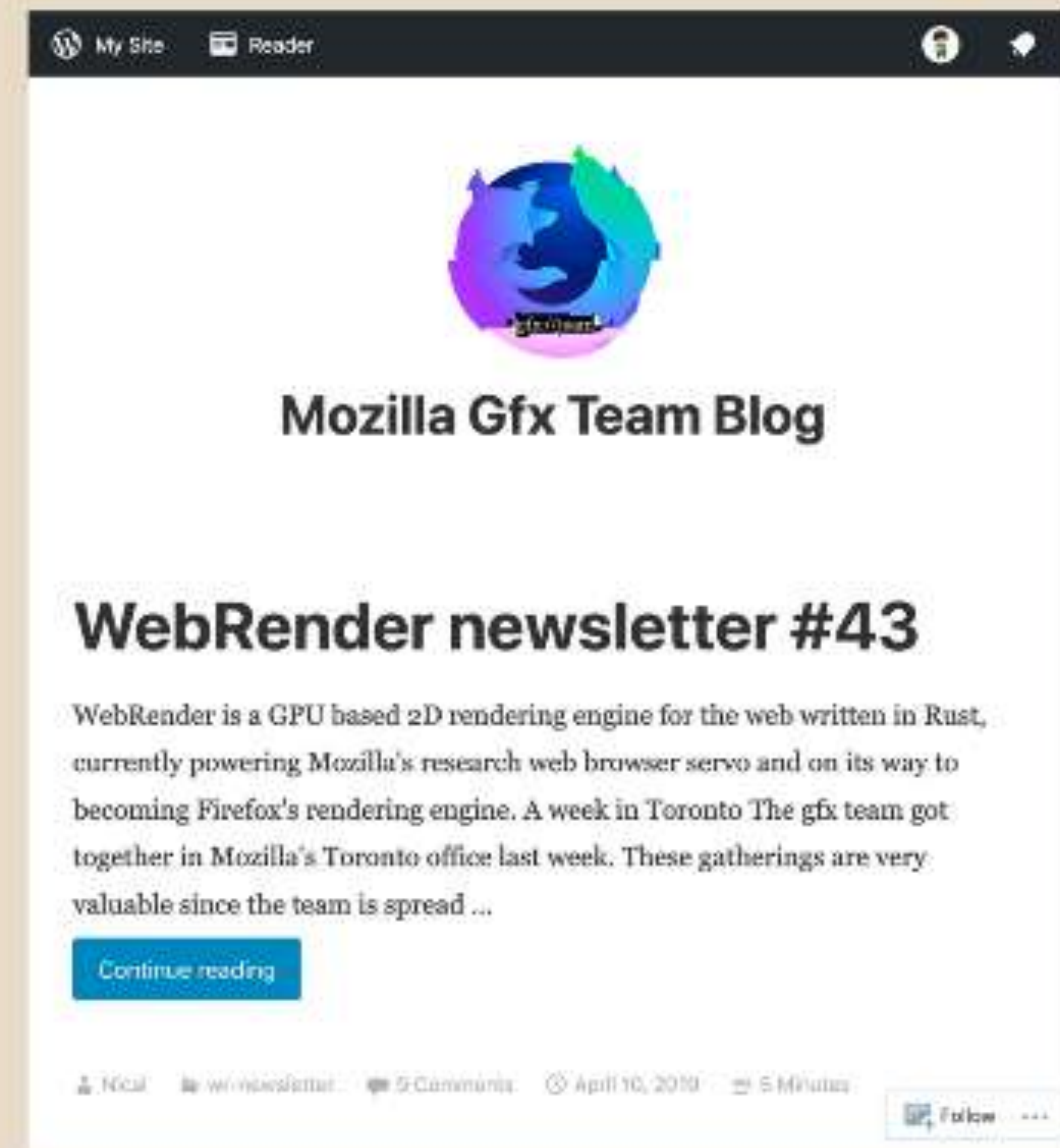
The whole web at maximum FPS: How WebRender gets rid of jank by Lin Clark



# WebRender (2/2)



<https://hacks.mozilla.org/2017/10/the-whole-web-at-maximum-fps-how-webrender-gets-rid-of-jank/>



<https://mozillagfx.wordpress.com/>



# Acknowledgements

🙏 Big thank you to these beautiful human beings who answered my noob questions 🙏



@bmeurer



@callahad



@slsoftworks



@linclark



@g33konaut



@mathias



@nicalsilva



@potch



# References

- Davis, W. (1986). *The Origins of Image Making*. Current Anthropology, 27(3), 193-215. doi:10.1086/203422
- [The GIF Is Dead. Long Live the GIF.](#)
- [Types of Bitmaps](#)
- [Why do we need JPG compression and how it's technically working?](#) by Steven Hansen
- [Progressive JPEGs and green Martians](#) by Jon Sneyers
- [Finally understanding JPG](#) by Christoph Erdmann
- [How JPG Works, How PNG Works, Reducing PNG file Size](#) by Colt McAnlis
- [Thoughts on a GIF-replacement file format](#)
- [Quantum Up Close: What is a browser engine?](#) by Matt "Potch" Claypotch
- [Let's build a browser engine!](#) by Matt Brubeck
- [On rendering engines and graphic libraries](#) by Kilian Valkhof
- [Following up on the 2d graphics in Rust discussion](#) by Nicolas Silva
- [Introduction to WebRender – Part 1 – Browsers today](#) by Nicolas Silva
- [The whole web at maximum FPS: How WebRender gets rid of jank](#) by Lin Clark
- [Software vs. GPU rasterization in Chromium\\*](#) by Martina Kollarova
- [GPU Accelerated Compositing in Chrome](#) by Tom Wiltzius, Vangelis Kokkevis & the Chrome Graphics team



**This talk is dedicated to browser engineers everywhere.  
I owe my career to you.**



# Thank you!



<https://www.chenhuijing.com>



[@hj\\_chen](#)



[@hj\\_chen](#)



[@huijing](#)

Font used is [Signika](#) by [Anna Giedryś](#)