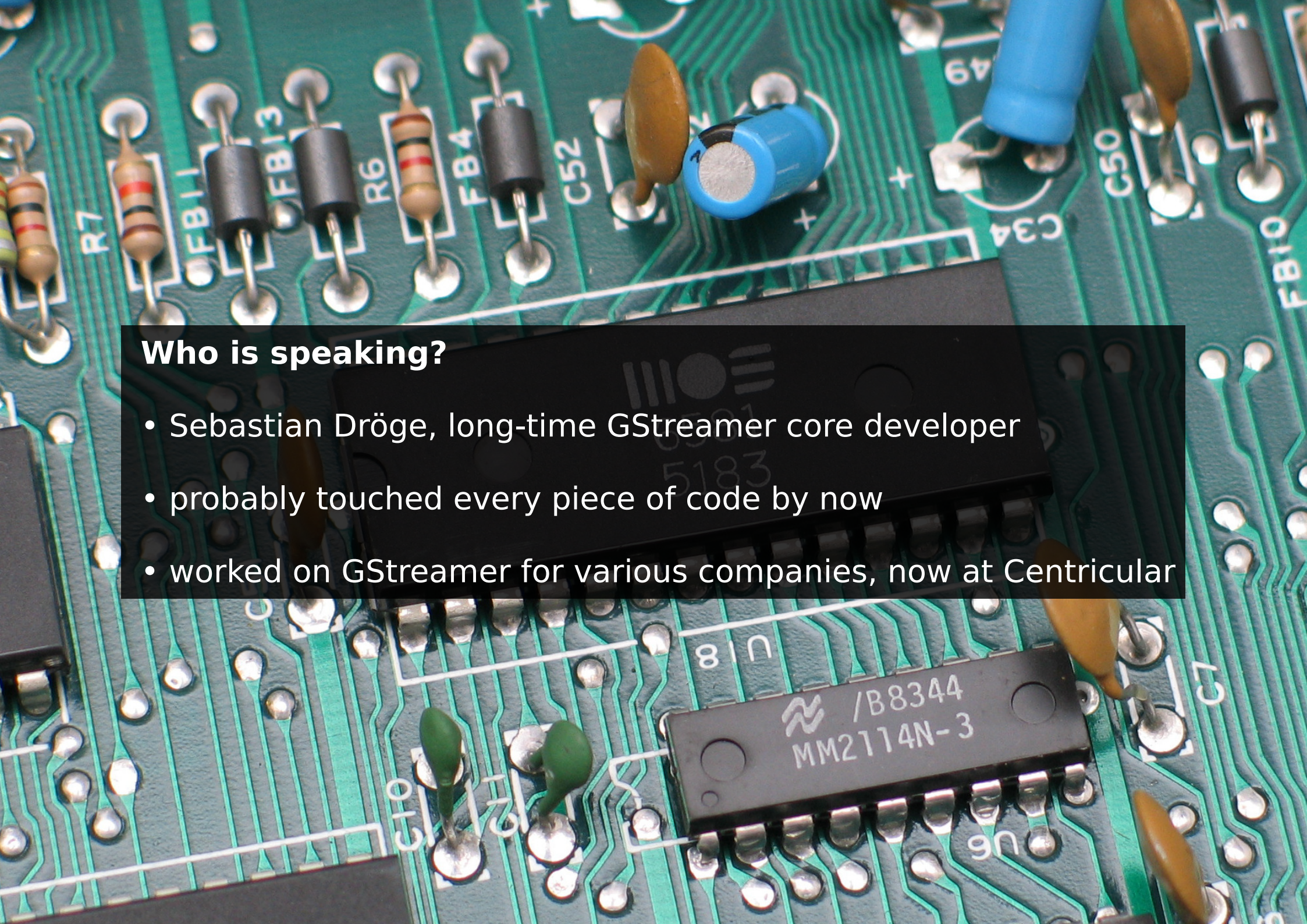
The never-ending story: GStreamer and hardware integration

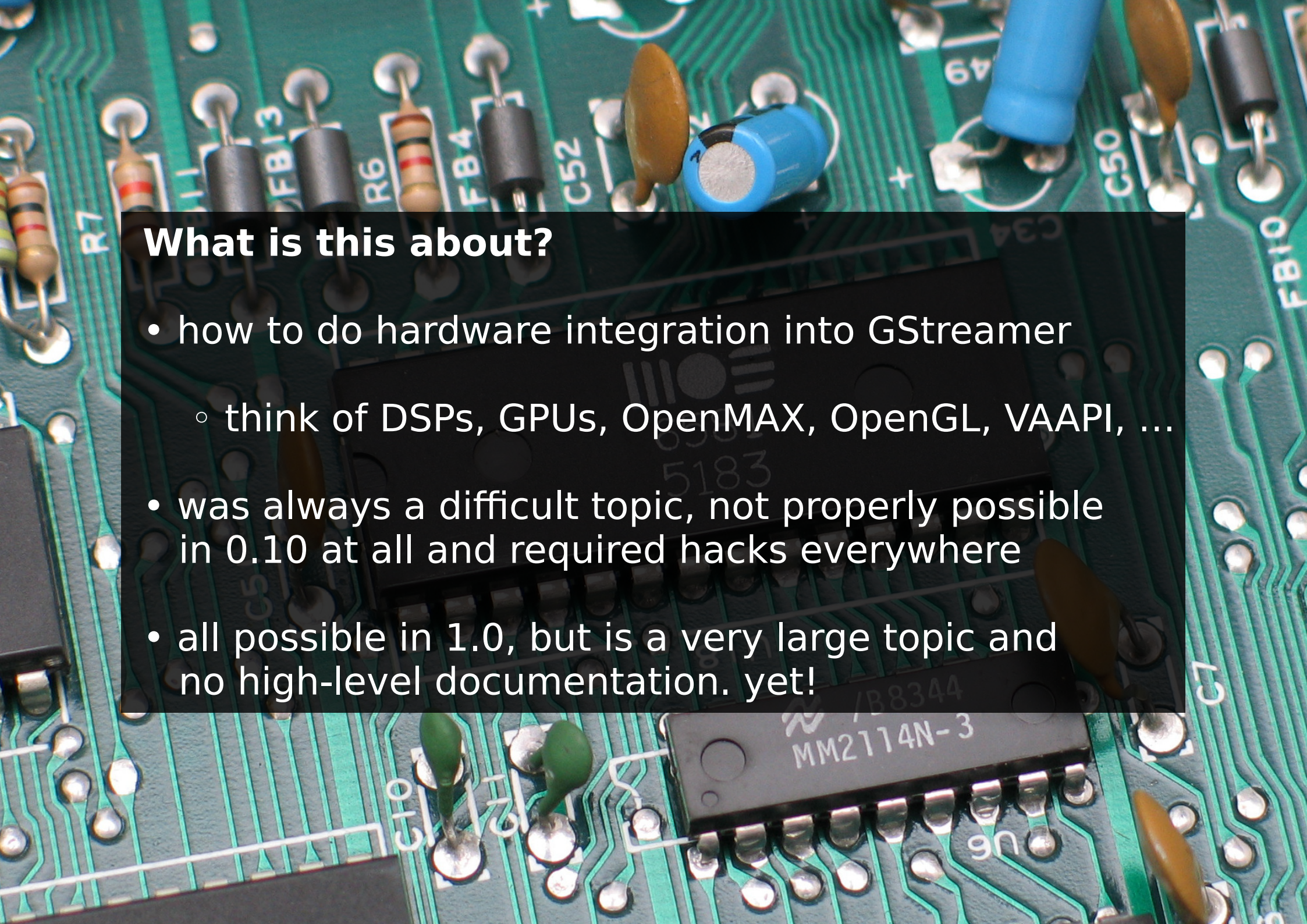
GStreamer Conference 2013, Edinburgh
22 October 2013

Sebastian Dröge <sebastian@centricular.com>
Centricular Ltd



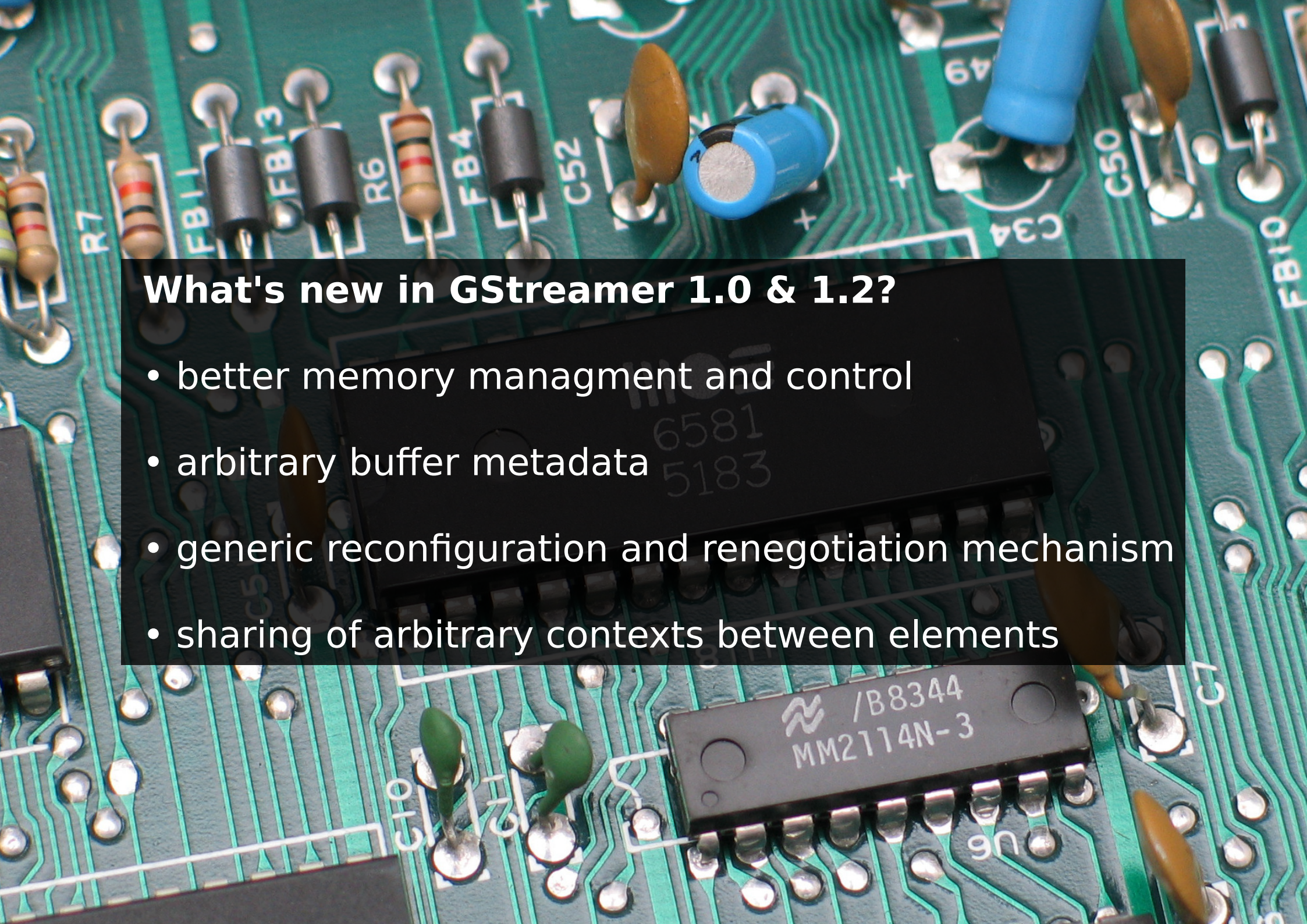
Who is speaking?

- Sebastian Dröge, long-time GStreamer core developer
- probably touched every piece of code by now
- worked on GStreamer for various companies, now at Centricular



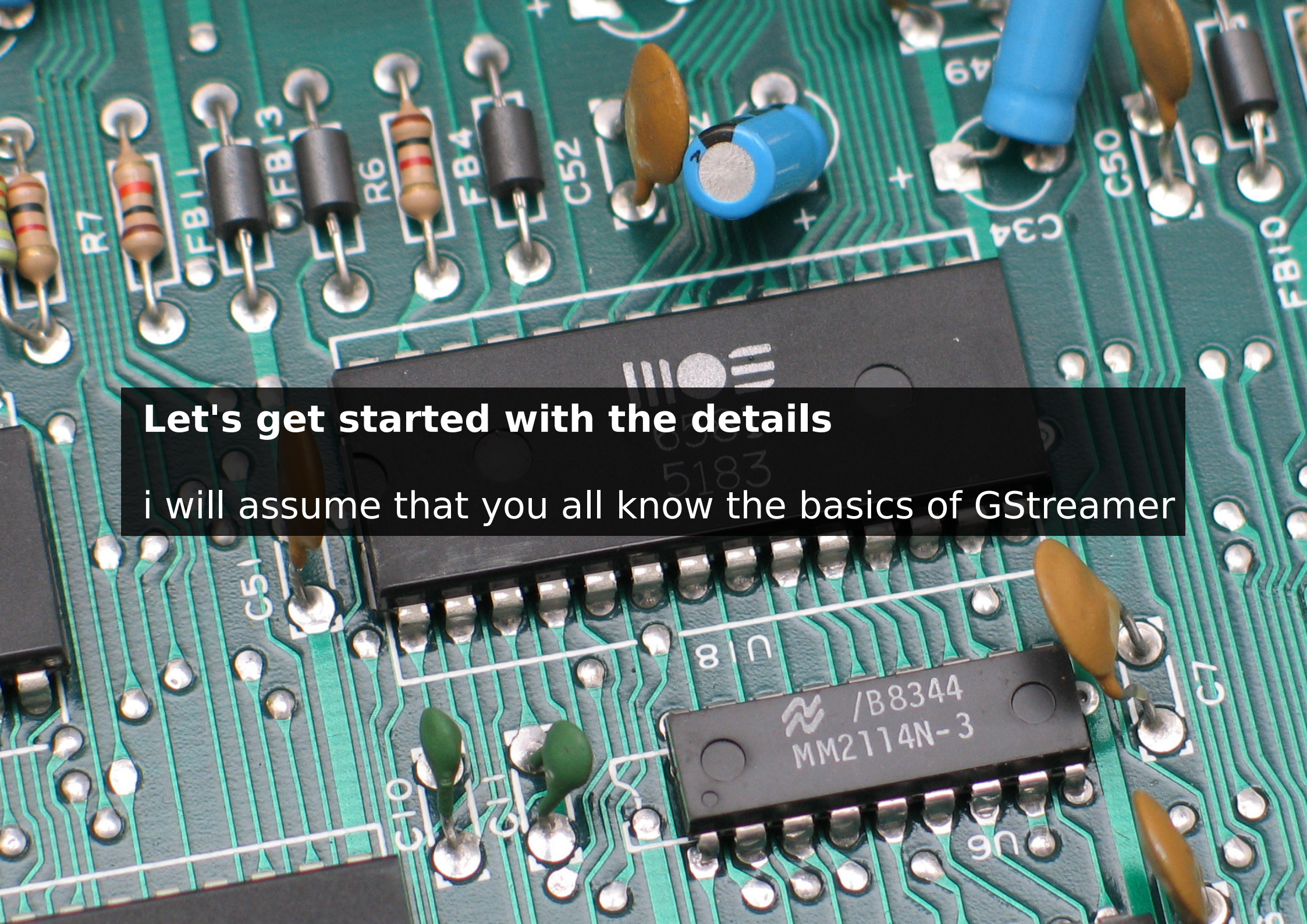
What is this about?

- how to do hardware integration into GStreamer
  - think of DSPs, GPUs, OpenMAX, OpenGL, VAAPI, ...
- was always a difficult topic, not properly possible in 0.10 at all and required hacks everywhere
- all possible in 1.0, but is a very large topic and no high-level documentation. yet!



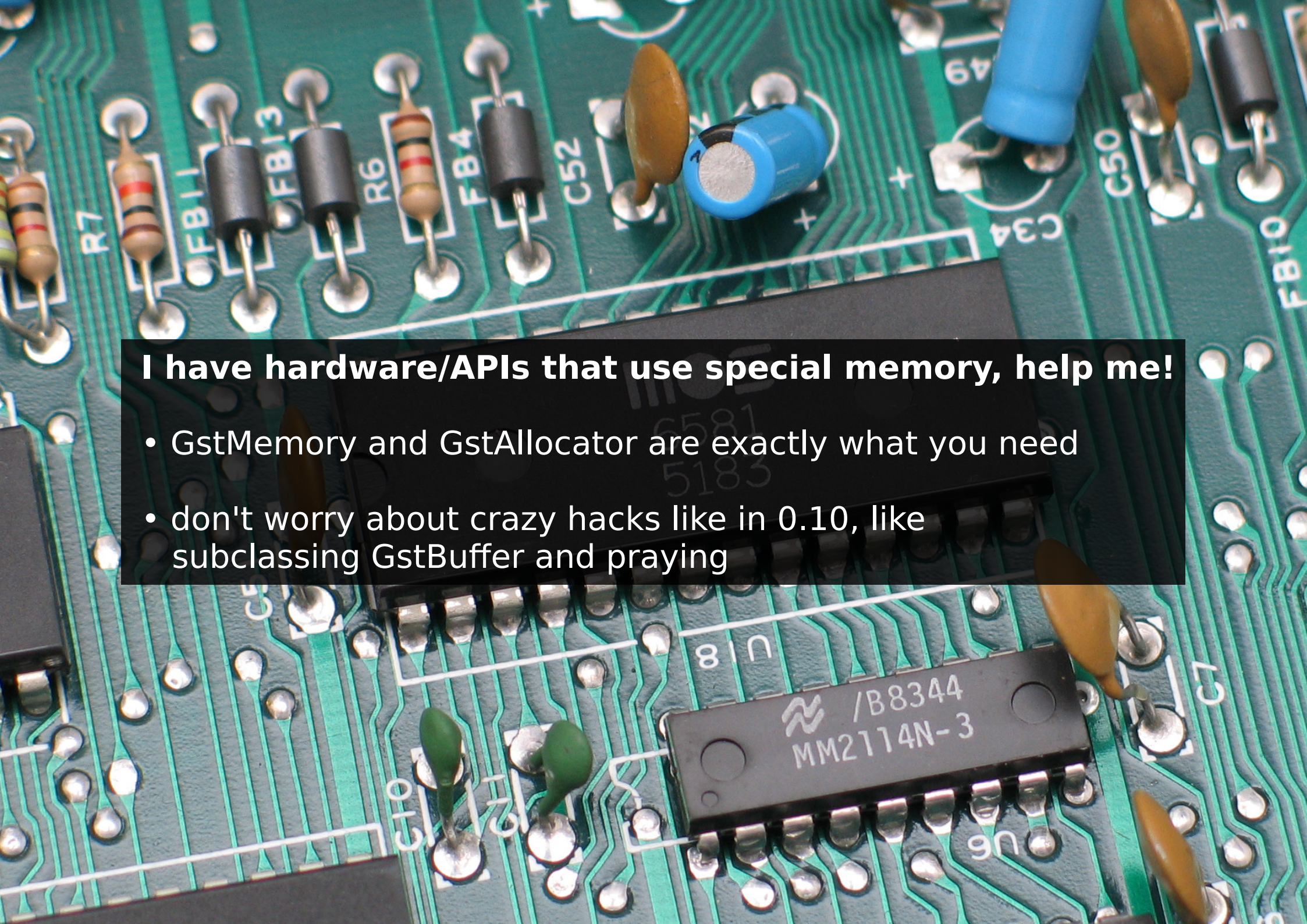
What's new in GStreamer 1.0 & 1.2?

- better memory managment and control
- arbitrary buffer metadata
- generic reconfiguration and renegotiation mechanism
- sharing of arbitrary contexts between elements



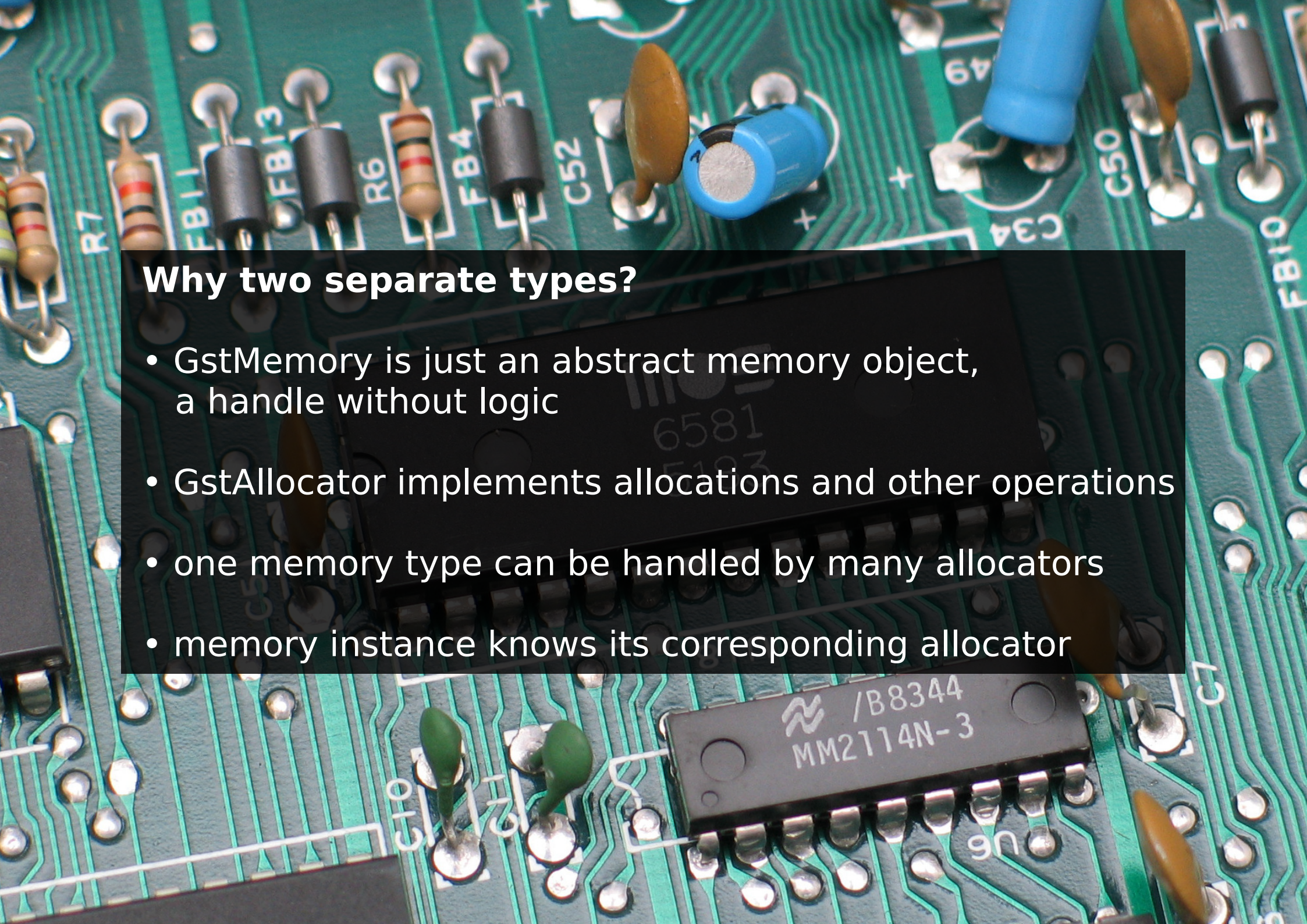
Let's get started with the details

i will assume that you all know the basics of GStreamer



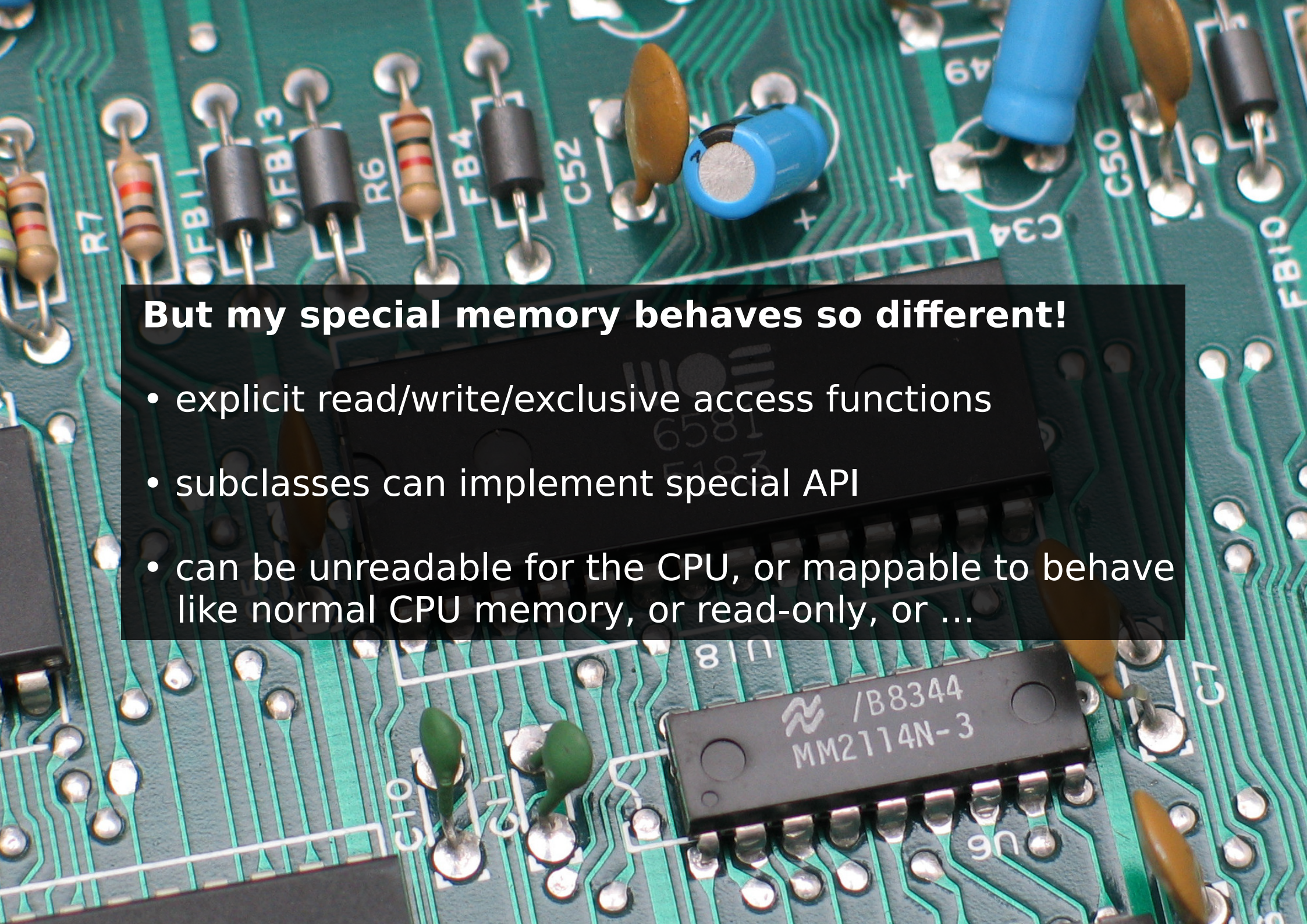
I have hardware/APIs that use special memory, help me!

- GstMemory and GstAllocator are exactly what you need
- don't worry about crazy hacks like in 0.10, like subclassing GstBuffer and praying



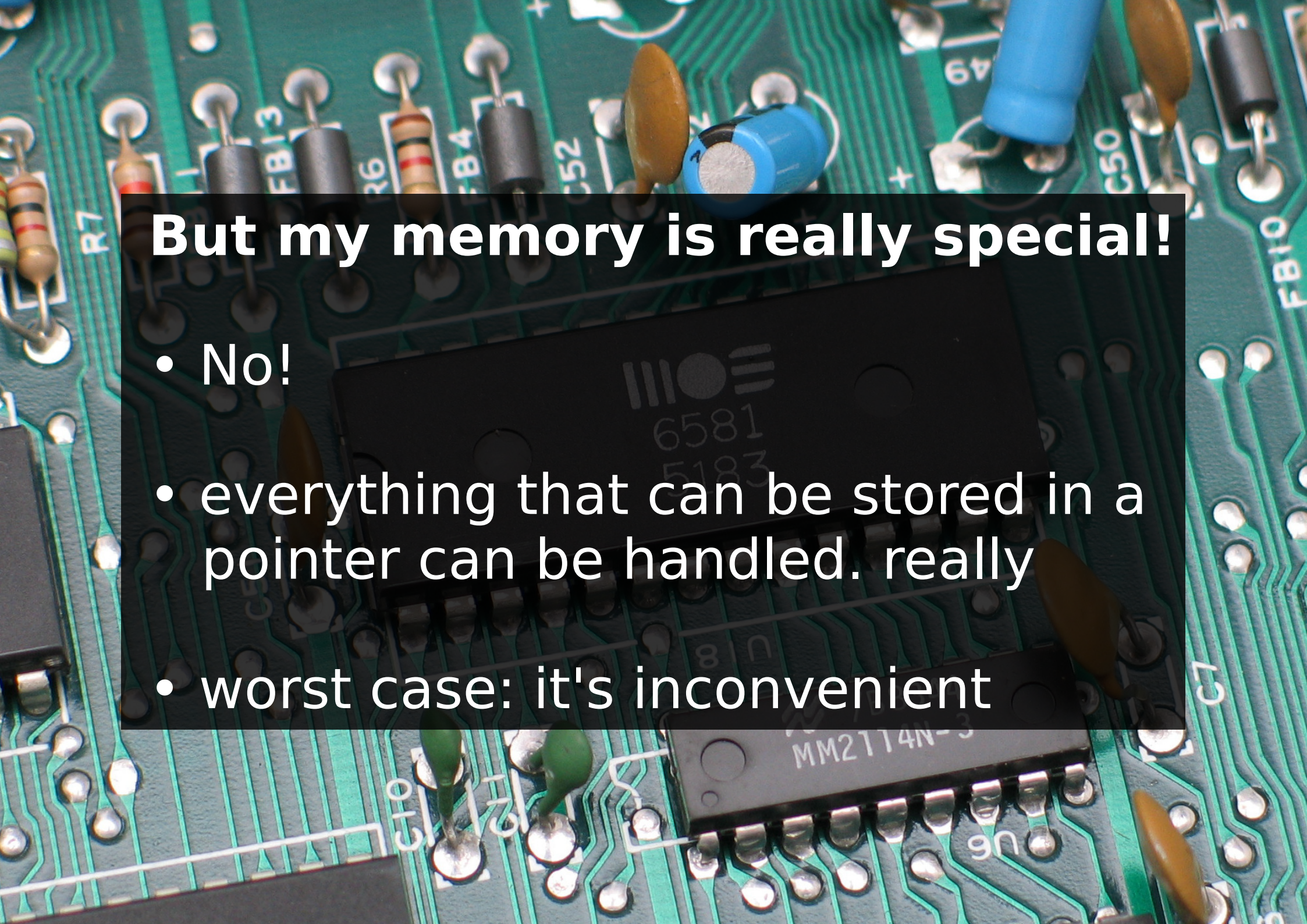
Why two separate types?

- GstMemory is just an abstract memory object, a handle without logic
- GstAllocator implements allocations and other operations
- one memory type can be handled by many allocators
- memory instance knows its corresponding allocator



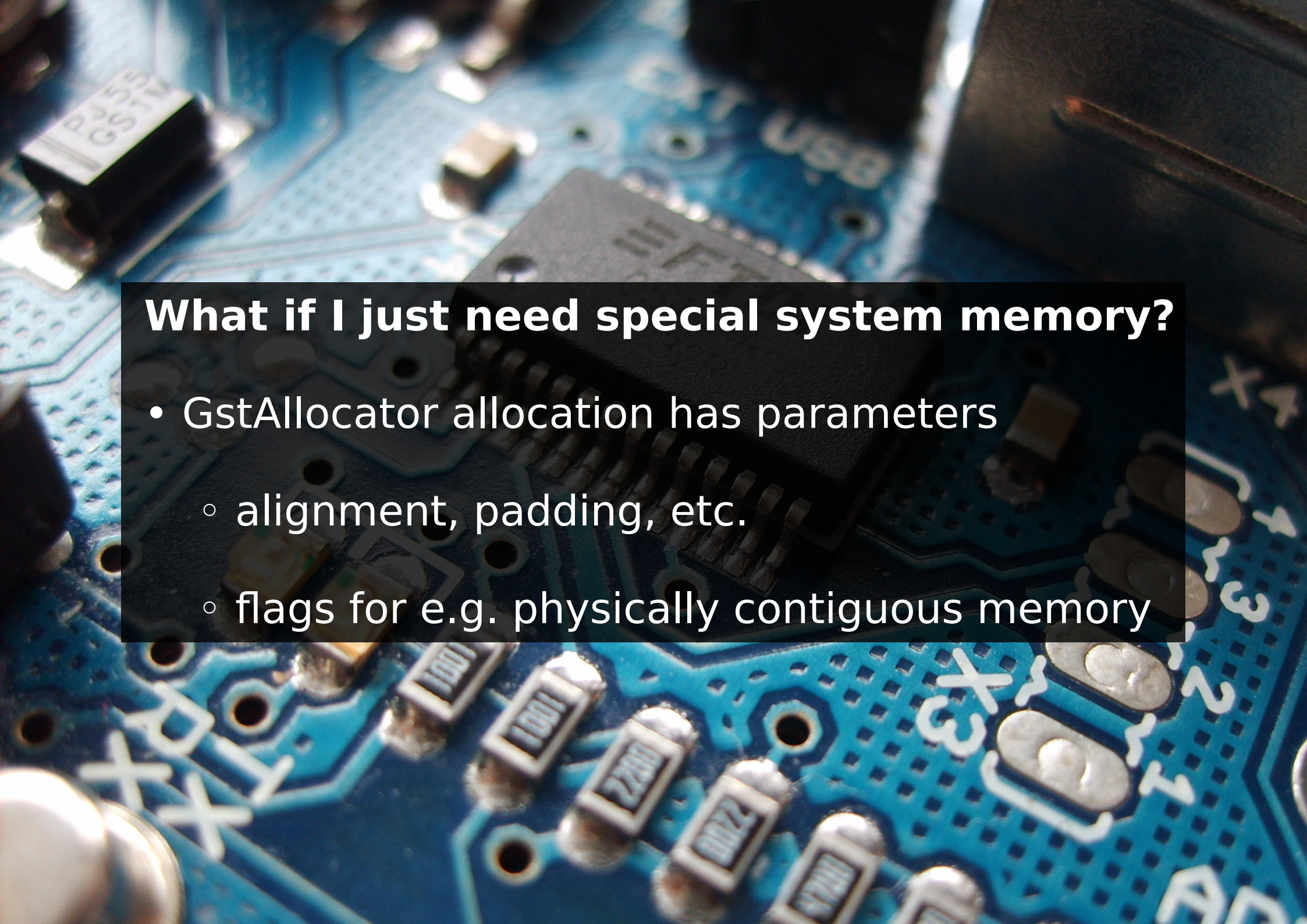
But my special memory behaves so different!

- explicit read/write/exclusive access functions
- subclasses can implement special API
- can be unreadable for the CPU, or mappable to behave like normal CPU memory, or read-only, or ...



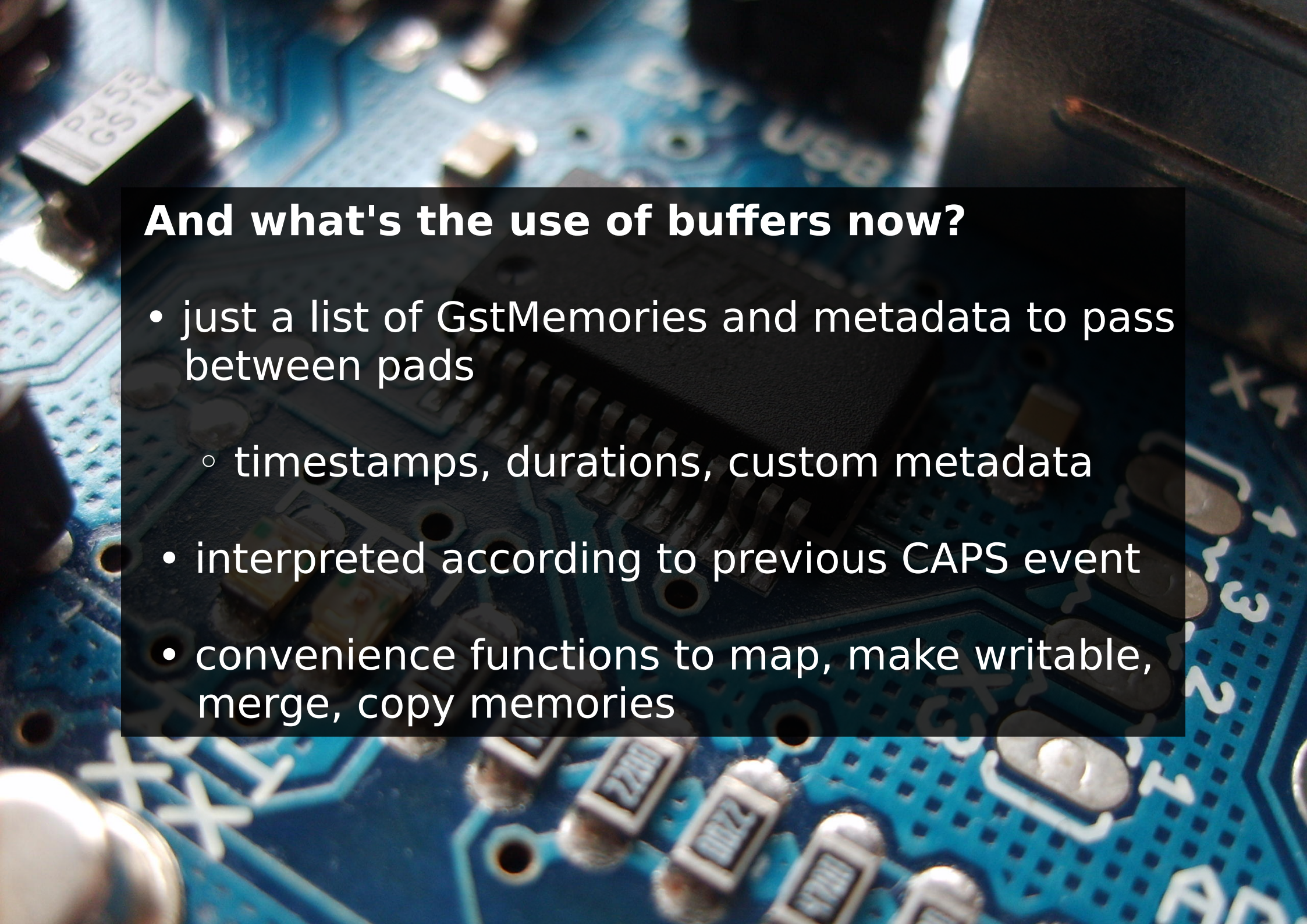
But my memory is really special!

- No!
- everything that can be stored in a pointer can be handled. really
- worst case: it's inconvenient



What if I just need special system memory?

- GstAllocator allocation has parameters
  - alignment, padding, etc.
  - flags for e.g. physically contiguous memory



And what's the use of buffers now?

- just a list of GstMemories and metadata to pass between pads
  - timestamps, durations, custom metadata
- interpreted according to previous CAPS event
- convenience functions to map, make writable, merge, copy memories

Ok, but I have to manage a pool of buffers!

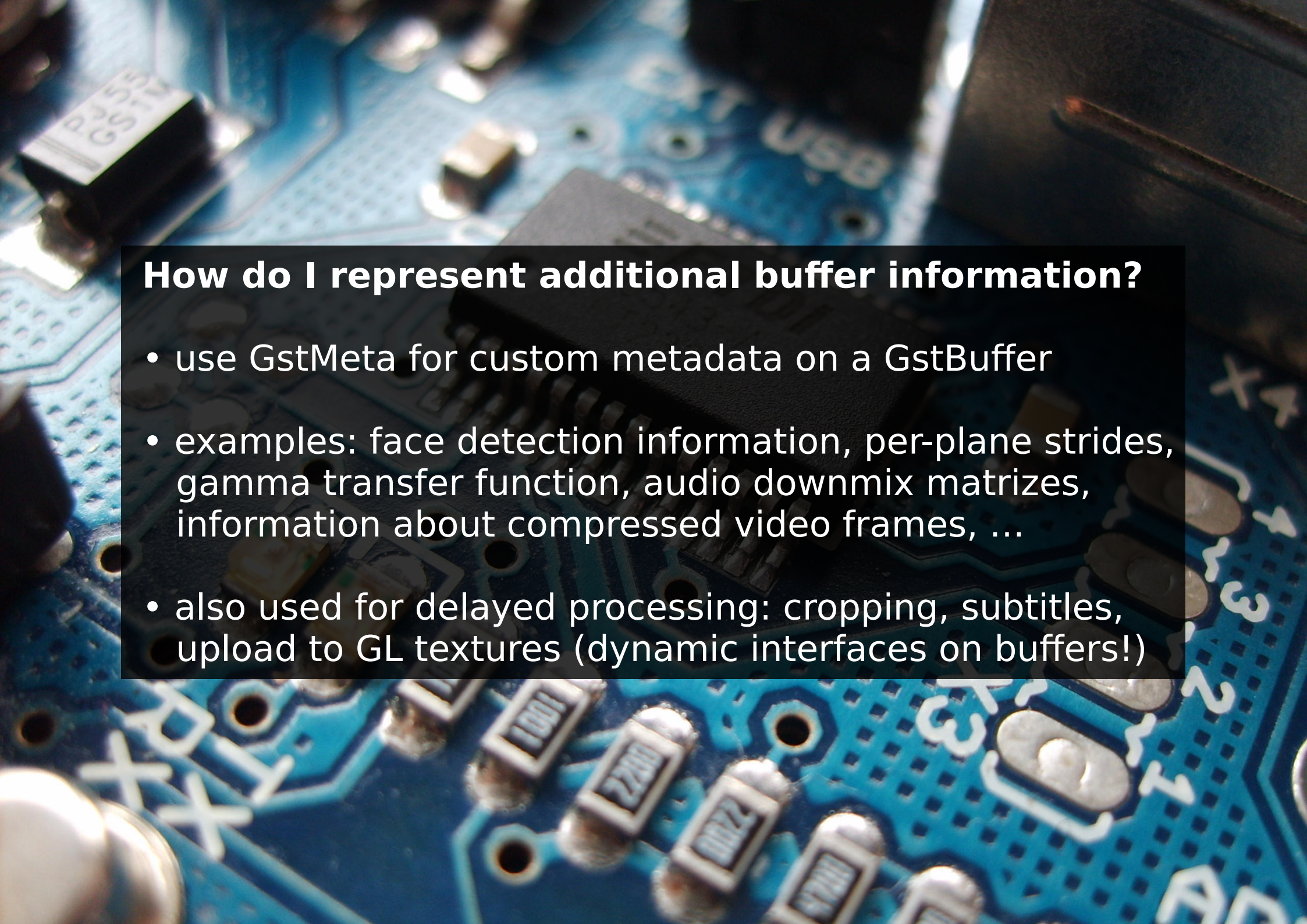
- useful if allocation is expensive or only have a limited pool of memory
- GstBufferPool provides base class with standard functionality
  - fixed/dynamic number of buffers, pre-allocate
  - acquiring buffer can block or fail
- allocates from a configurable GstAllocator

What about custom configuration?

- generic configuration interface
  - allow new allocations
  - min/max num. of buffers, allocation size, parameters, GstAllocator
  - extendable by subclass
- custom features can be queried

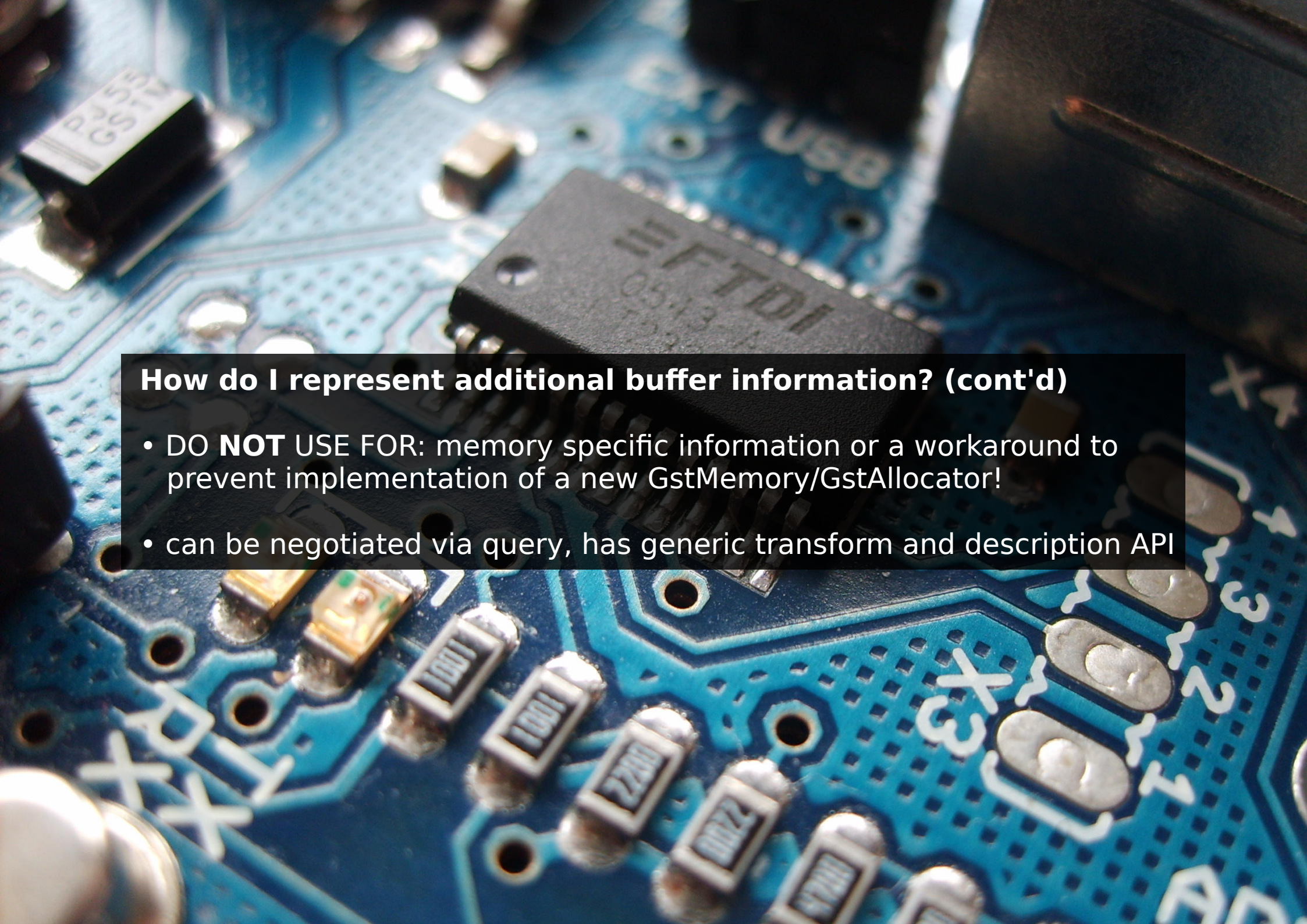
And raw video? It's so complicated!

- GstVideoBufferPool subclass
- supports video meta, cares for correct allocation size
- GstVideoAlign configuration for per-plane padding



How do I represent additional buffer information?

- use GstMeta for custom metadata on a GstBuffer
- examples: face detection information, per-plane strides, gamma transfer function, audio downmix matrizes, information about compressed video frames, ...
- also used for delayed processing: cropping, subtitles, upload to GL textures (dynamic interfaces on buffers!)

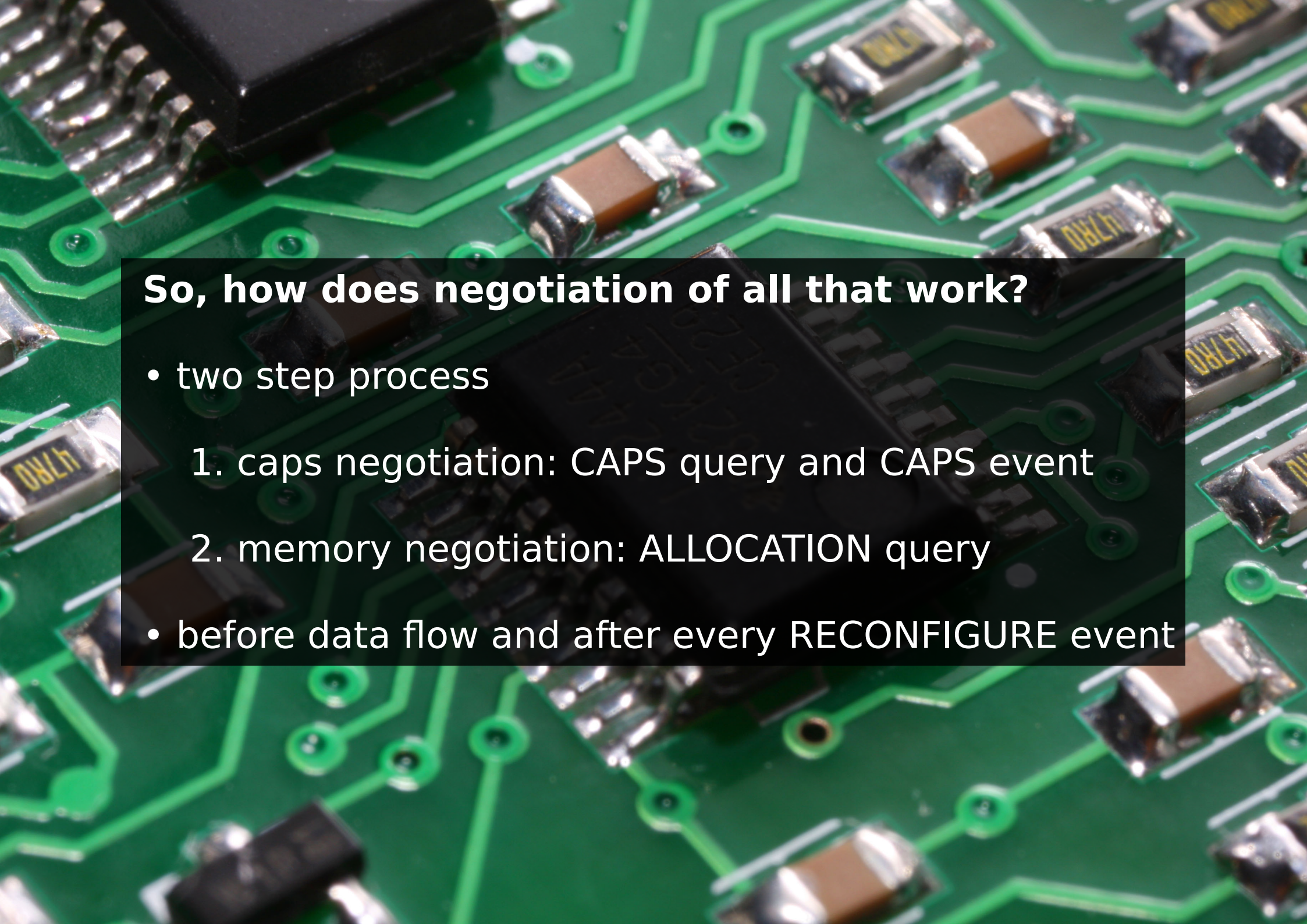


How do I represent additional buffer information? (cont'd)

- DO **NOT** USE FOR: memory specific information or a workaround to prevent implementation of a new GstMemory/GstAllocator!
- can be negotiated via query, has generic transform and description API

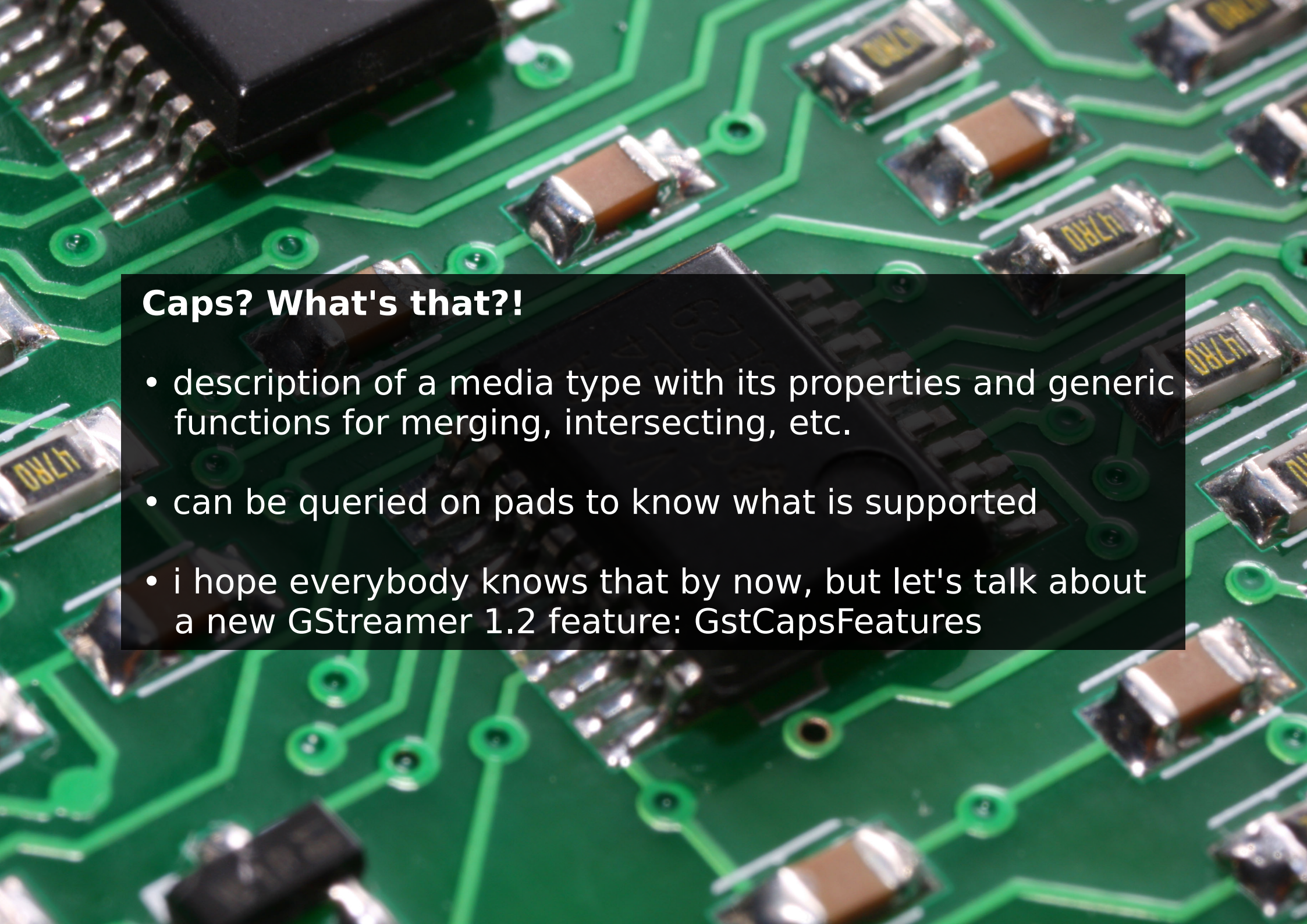
But I don't want to define everything myself!

- many common GstMeta provided by GStreamer libraries
- GstVideoMeta, GstVideoCropMeta, GstVideoGLTextureUploadMeta, GstVideoOverlayMeta, GstVideoRegionOfInterestMeta
- GstAudioDownmixMeta
- GstMPegVideoMeta, others for compressed formats
- to be continued!



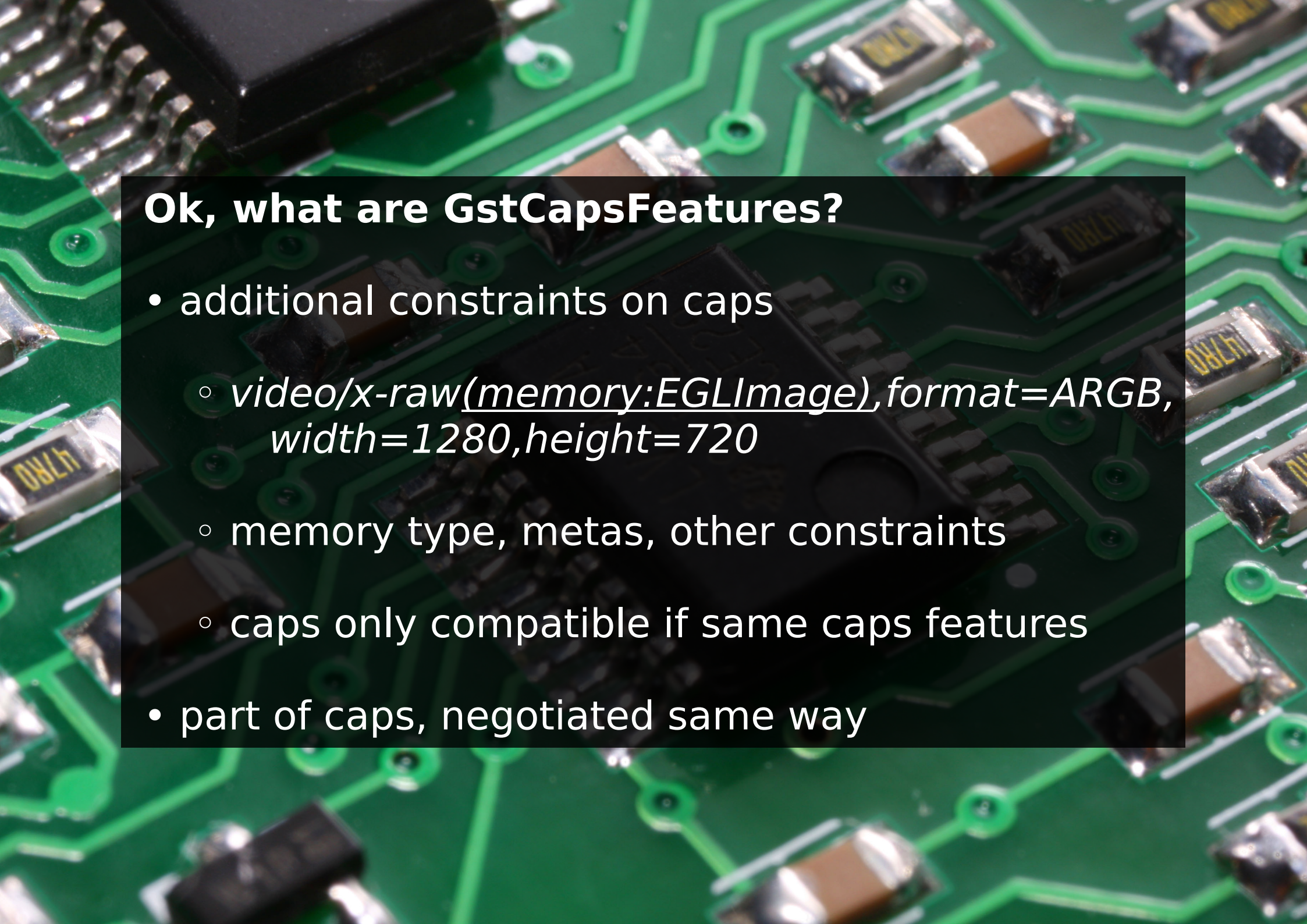
So, how does negotiation of all that work?

- two step process
  1. caps negotiation: CAPS query and CAPS event
  2. memory negotiation: ALLOCATION query
- before data flow and after every RECONFIGURE event



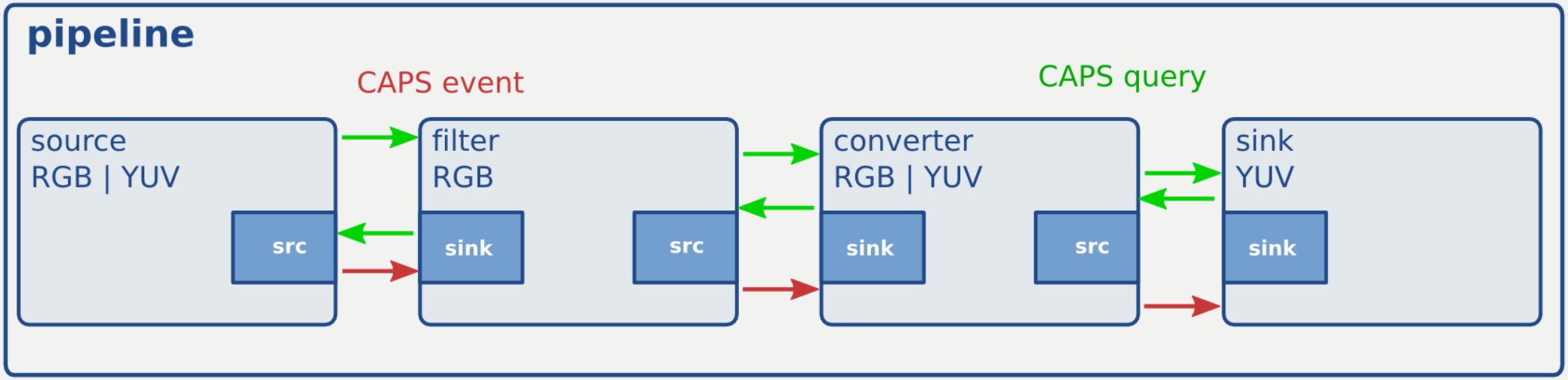
Caps? What's that?!

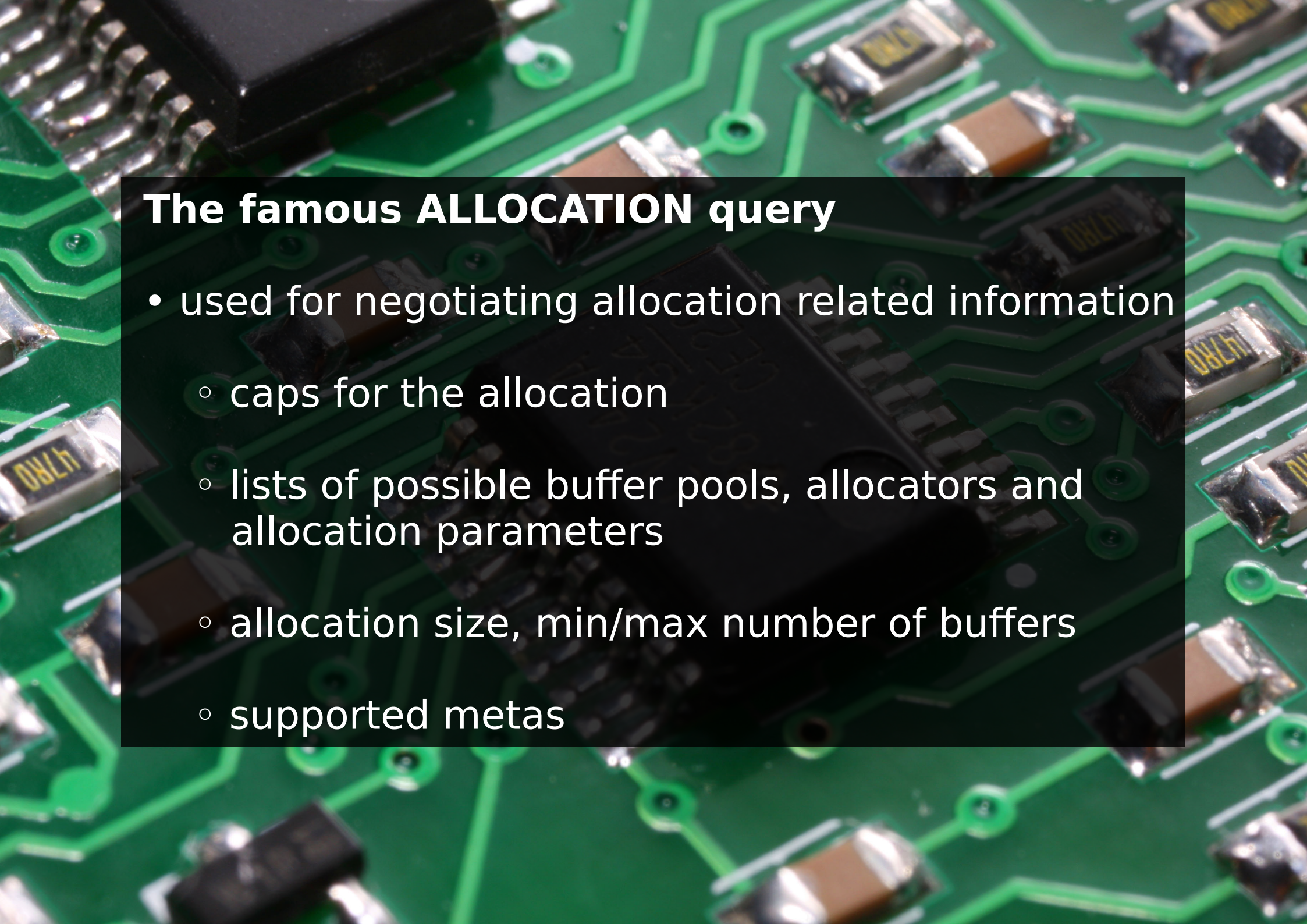
- description of a media type with its properties and generic functions for merging, intersecting, etc.
- can be queried on pads to know what is supported
- i hope everybody knows that by now, but let's talk about a new GStreamer 1.2 feature: GstCapsFeatures



Ok, what are GstCapsFeatures?

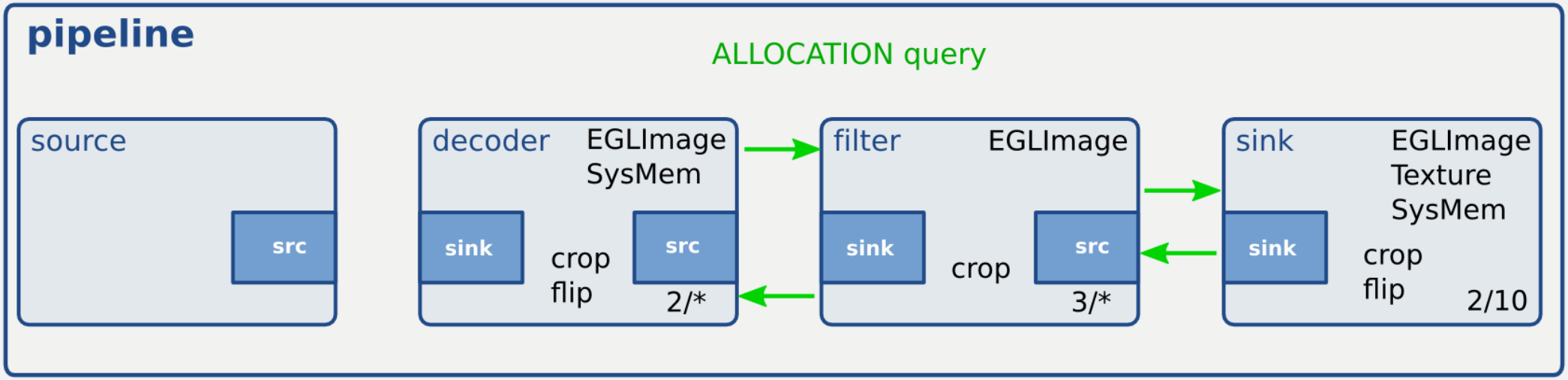
- additional constraints on caps
  - *video/x-raw(memory:EGLImage),format=ARGB, width=1280,height=720*
  - memory type, metas, other constraints
  - caps only compatible if same caps features
- part of caps, negotiated same way

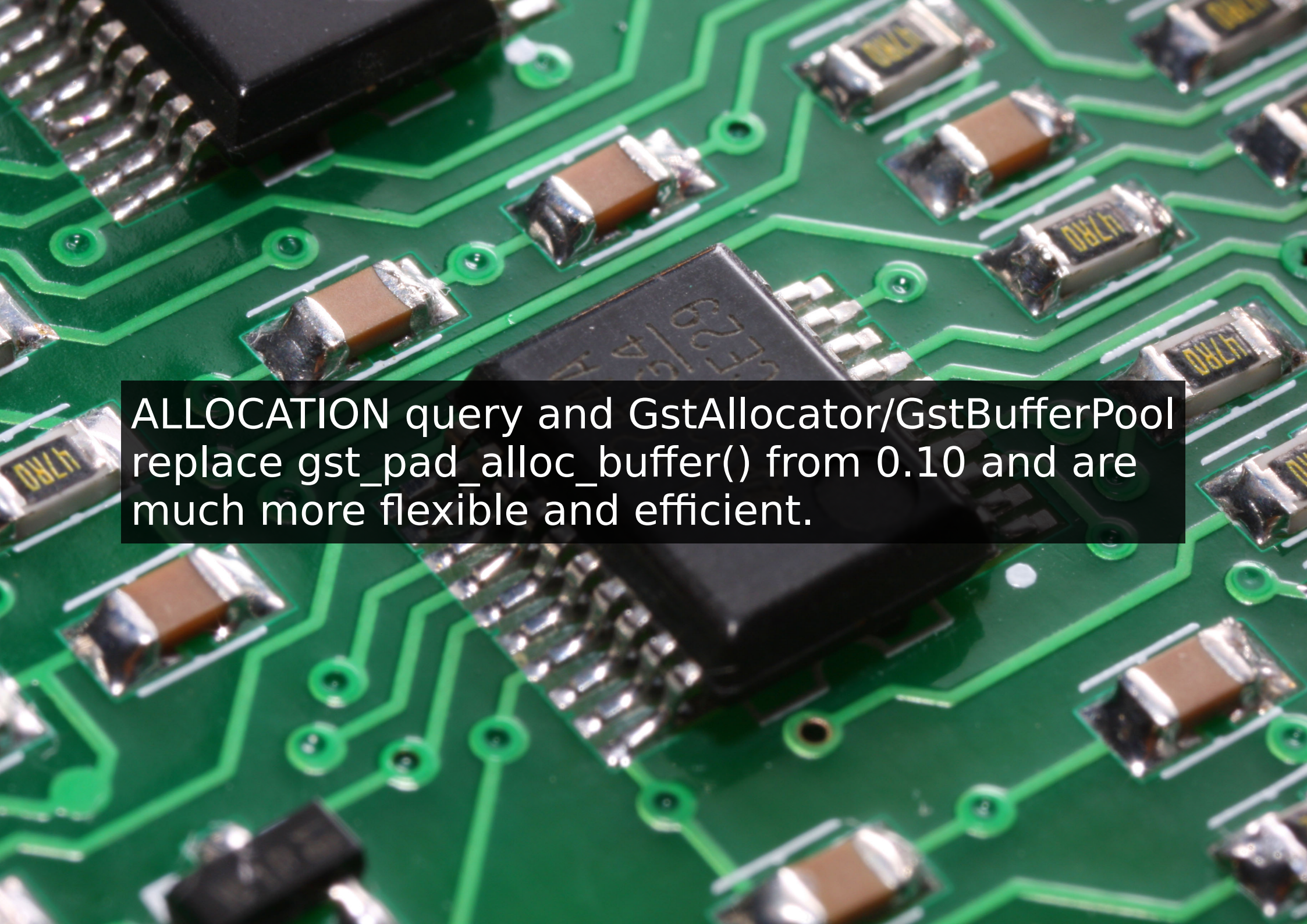

pipeline
CAPS event
CAPS query
source
RGB | YUV
src
filter
RGB
sink
src
converter
RGB | YUV
sink
src
sink
YUV
sink



The famous ALLOCATION query

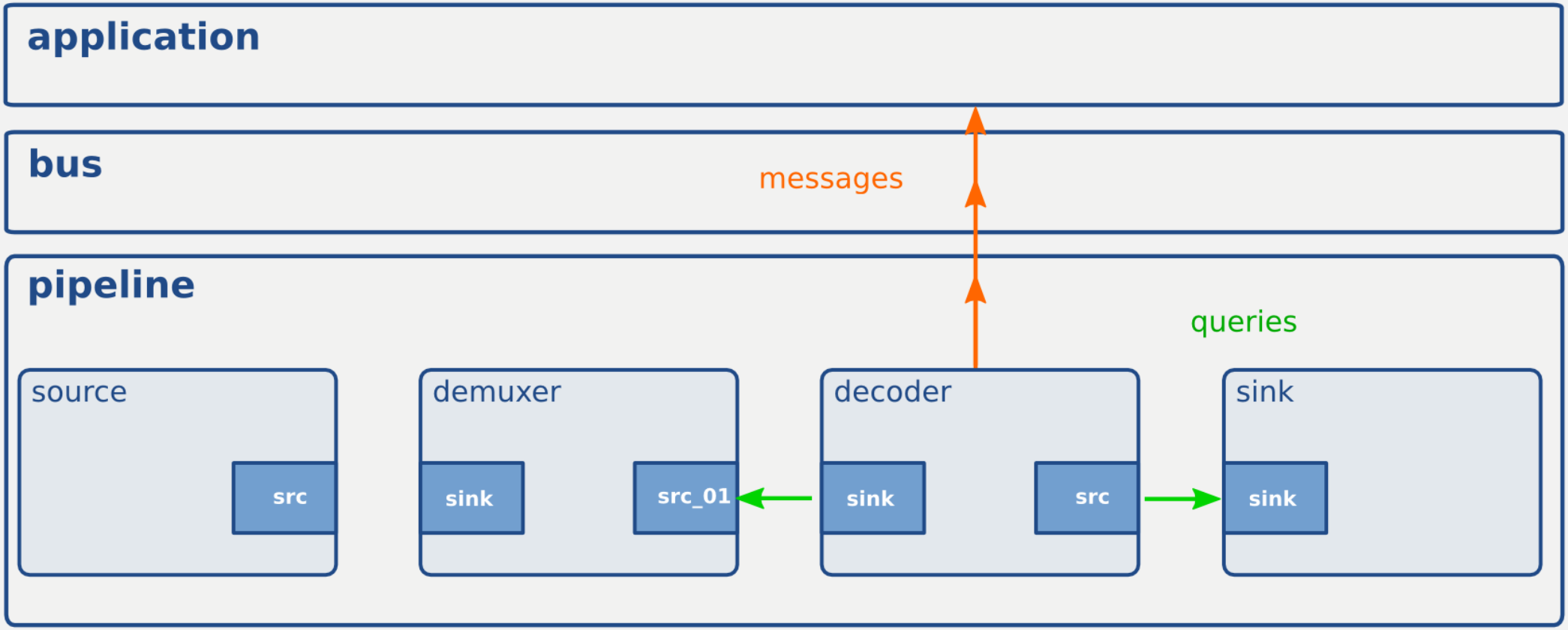
- used for negotiating allocation related information
  - caps for the allocation
  - lists of possible buffer pools, allocators and allocation parameters
  - allocation size, min/max number of buffers
  - supported metas

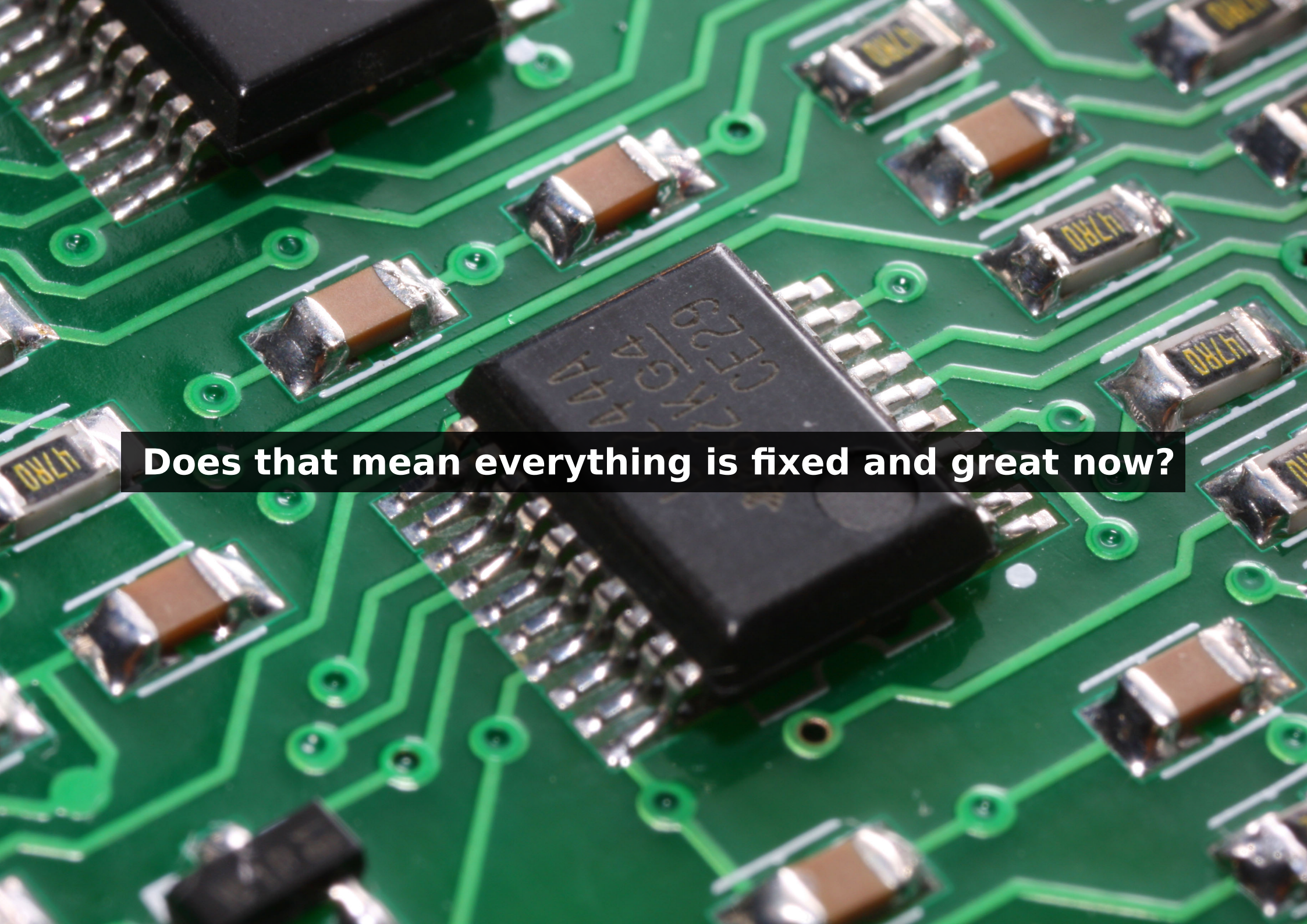

pipeline
ALLOCATION query
source
src
decoder
EGLImage
SysMem
sink
crop
flip
src
2/*
filter
EGLImage
sink
crop
src
3/*
sink
EGLImage
Texture
SysMem
sink
crop
flip
2/10

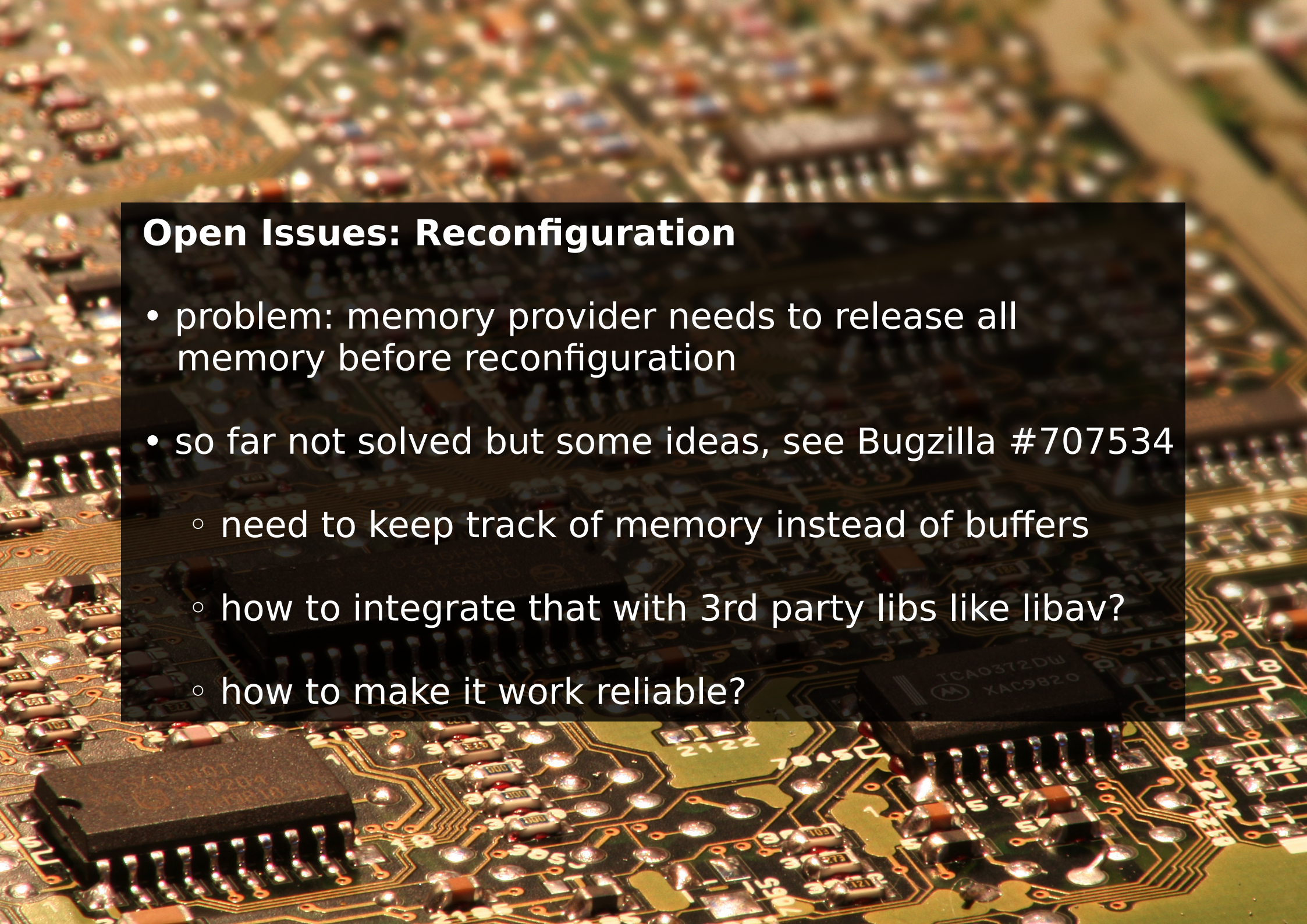

ALLOCATION query and GstAllocator/GstBufferPool replace gst_pad_alloc_buffer() from 0.10 and are much more flexible and efficient.

But my elements need to share some common context!

- use GstContext and related queries/messages for this
- examples: VADisplay, OpenGL context, EGLDisplay, HTTP session, ...
- queries for getting local contexts, messages for global contexts
- bins are caching and propagating contexts based on messages
- so how does it work in practice?

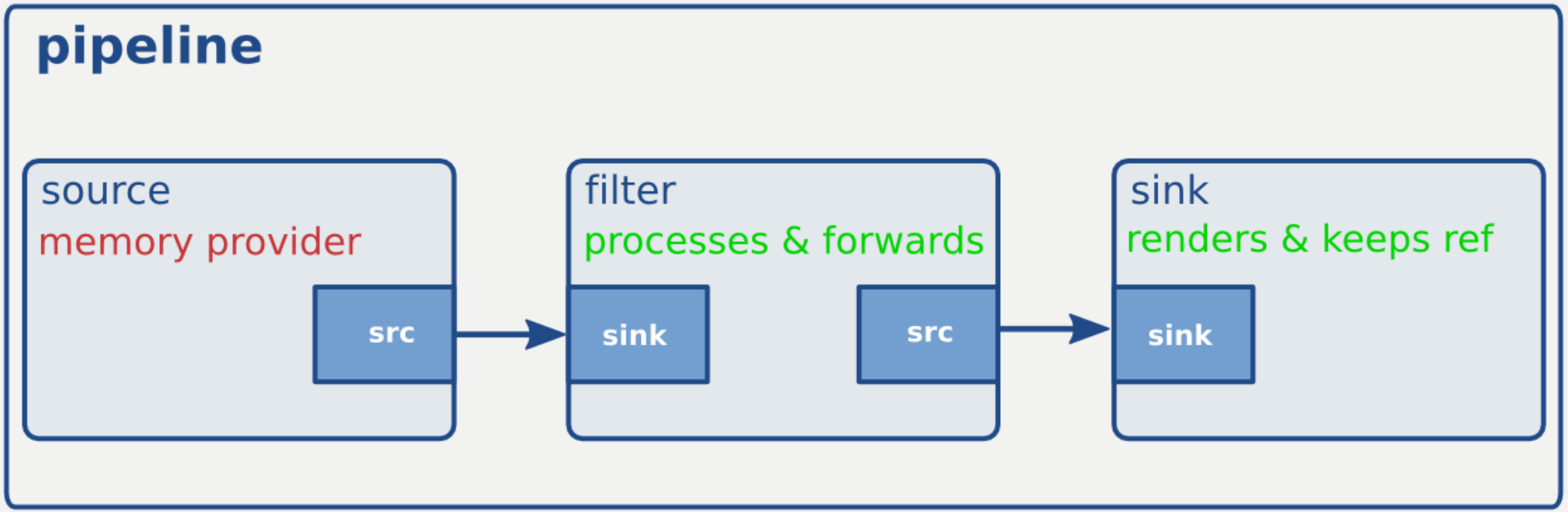

application
bus
messages
pipeline
queries
source
src
demuxer
sink
src_01
decoder
sink
src
sink
sink

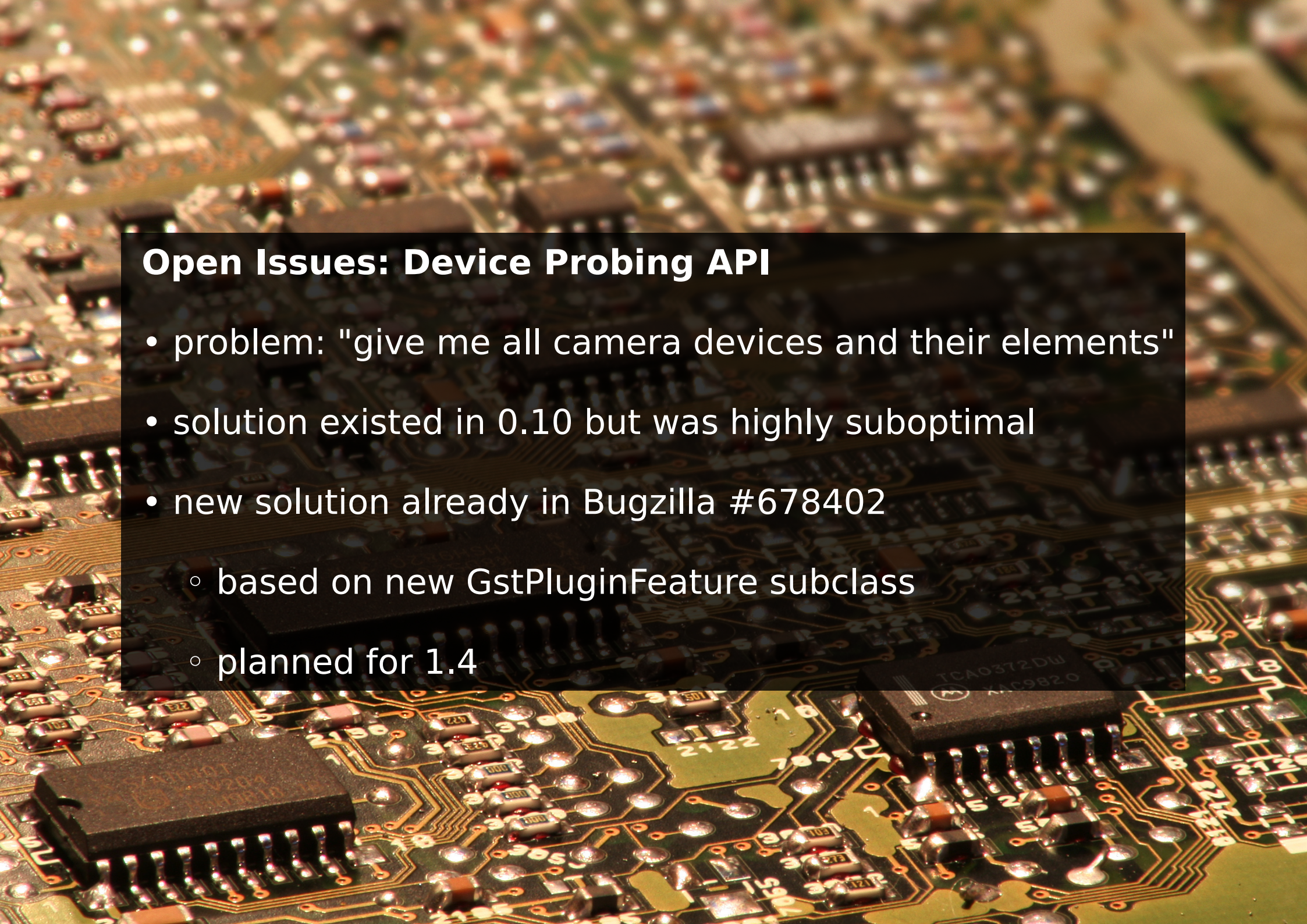

Does that mean everything is fixed and great now?



Open Issues: Reconfiguration

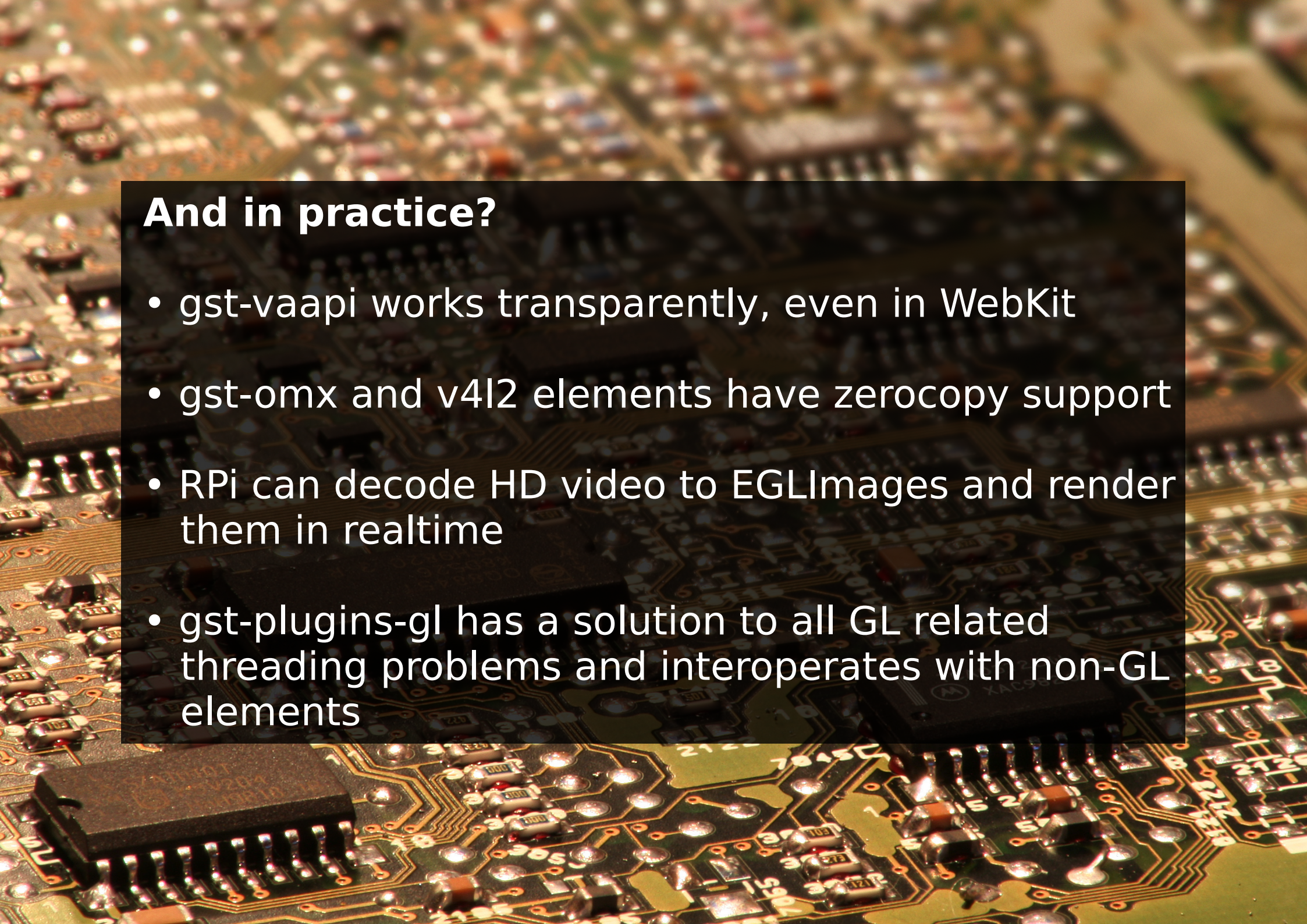
- problem: memory provider needs to release all memory before reconfiguration
- so far not solved but some ideas, see Bugzilla #707534
  - need to keep track of memory instead of buffers
  - how to integrate that with 3rd party libs like libav?
  - how to make it work reliable?


pipeline
source
memory provider
src
filter
processes & forwards
sink
src
sink
renders & keeps ref
sink



Open Issues: Device Probing API

- problem: "give me all camera devices and their elements"
- solution existed in 0.10 but was highly suboptimal
- new solution already in Bugzilla #678402
  - based on new GstPluginFeature subclass
  - planned for 1.4



And in practice?

- gst-vaapi works transparently, even in WebKit
- gst-omx and v4l2 elements have zerocopy support
- RPi can decode HD video to EGLImages and render them in realtime
- gst-plugins-gl has a solution to all GL related threading problems and interoperates with non-GL elements

And in practice? (cont'd)

let me repeat: completely transparent, elements or applications don't have to know

... and all this without 0.10-style hacks!

Questions?

also feel free to talk to me later or write a mail

sebastian@centricular.com

summary of this talk will be on my personal blog:
http://coaxion.net/blog

Thank You!

Pictures

Bulky City by Peter Kemmer, CC BY-SA-NC 2.0
MOS 6581 sound chip from C64 by Christian Taube, CC BY-SA 2.5
Arduino FTDI chip by Dusty Dingo, Public Domain
JTAG board 1 by Andrew Magil, CC BY 2.0
Silicon City by Tomizak, CC BY-ND 2.0