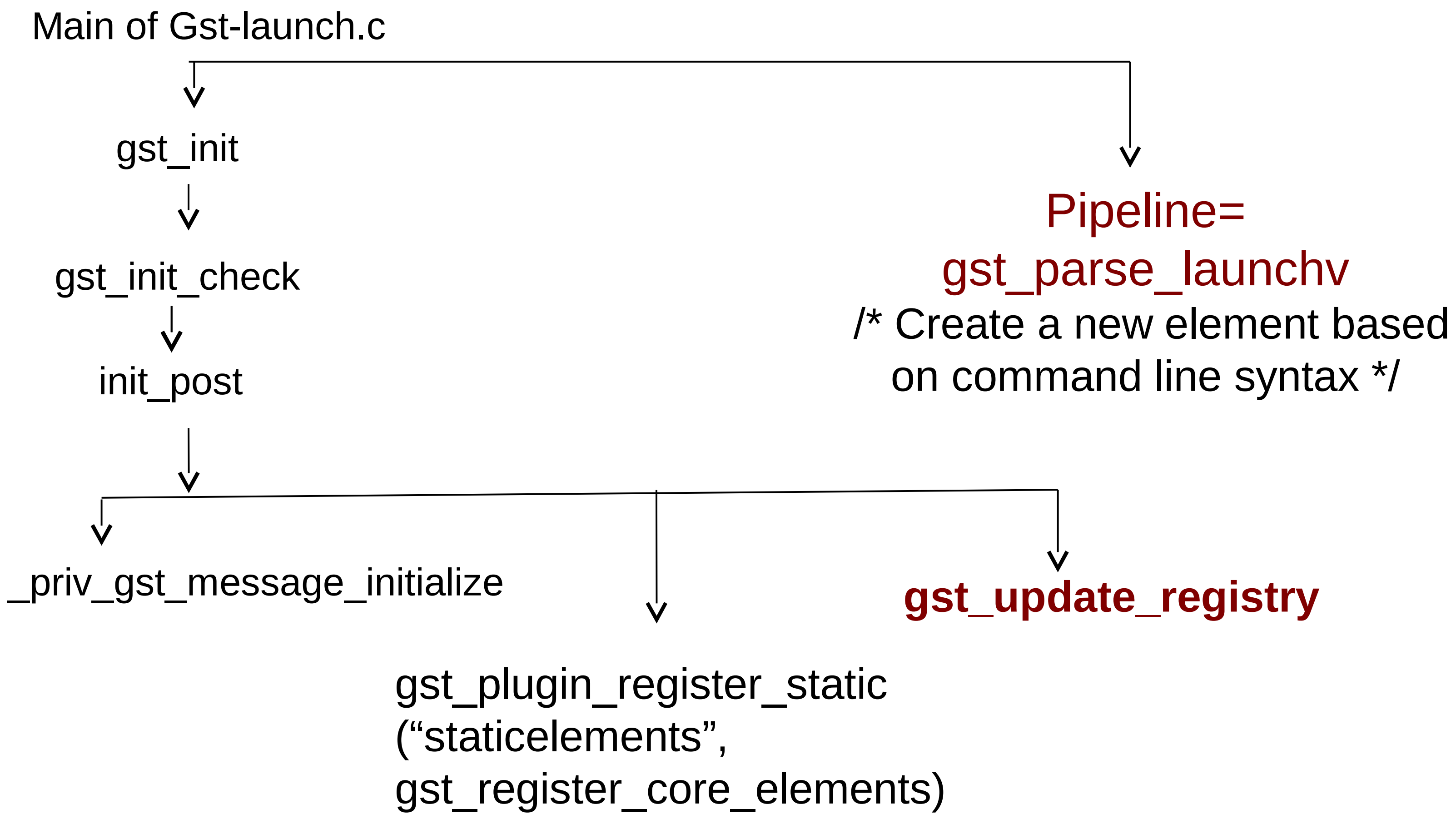
Case for this study:
gst-launch-1.0 playbin xxx.ts

1, Load registry from binary plugin feature structures


Main of Gst-launch.c
gst_init
gst_init_check
init_post
_priv_gst_message_initialize
gst_plugin_register_static
(“staticelements”,
gst_register_core_elements)
gst_update_registry
Pipeline=
gst_parse_launchv
/* Create a new element based
on command line syntax */

gst_update_registry

↓

gst_registry_chunks_load_feature

/* Make a new GstPluginFeature from current binary plugin feature structure */

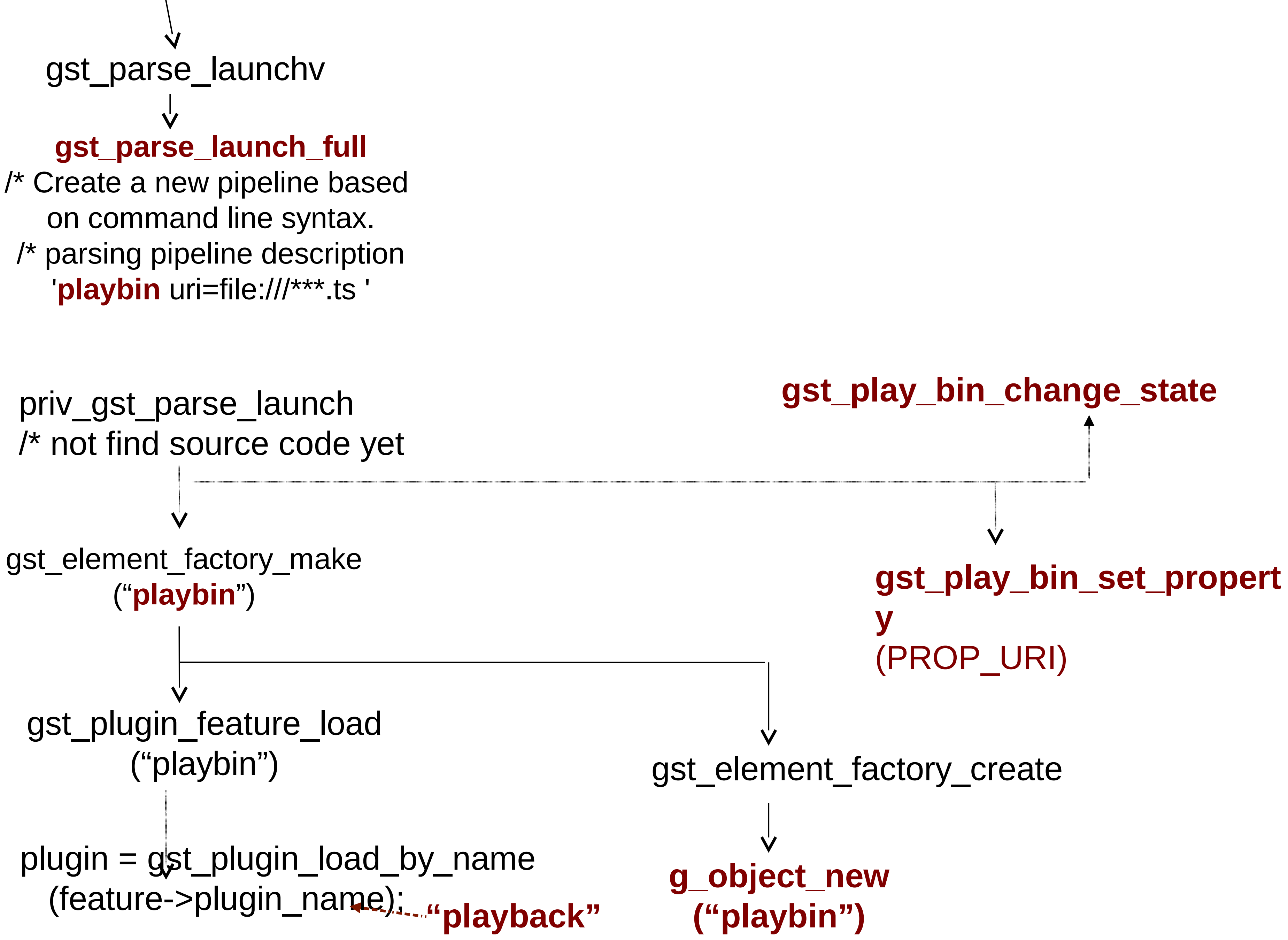
↓

Plugin 'playback' feature 'playbin'

↓

Added feature playbin for plugin playback

2, Initializing actions by analyzing argv – playbin & uri


gst_parse_launchv

gst_parse_launch_full
/* Create a new pipeline based on command line syntax.
/* parsing pipeline description
'**playbin** uri=file:///***.ts '

priv_gst_parse_launch
/* not find source code yet

gst_play_bin_change_state

gst_element_factory_make
("**playbin**")

gst_play_bin_set_propert y
(PROP_URI)

gst_plugin_feature_load
("playbin")

gst_element_factory_create

plugin = gst_plugin_load_by_name
(feature->plugin_name);

"playback"

g_object_new
("playbin")

gst_plugin_feature_load
("playbin")

$\downarrow$

if("playbin" not loaded)
gst_plugin_load_by_name
("playback")

$\downarrow$

loading plugin playback from file
libgstplayback.so

$\longrightarrow$

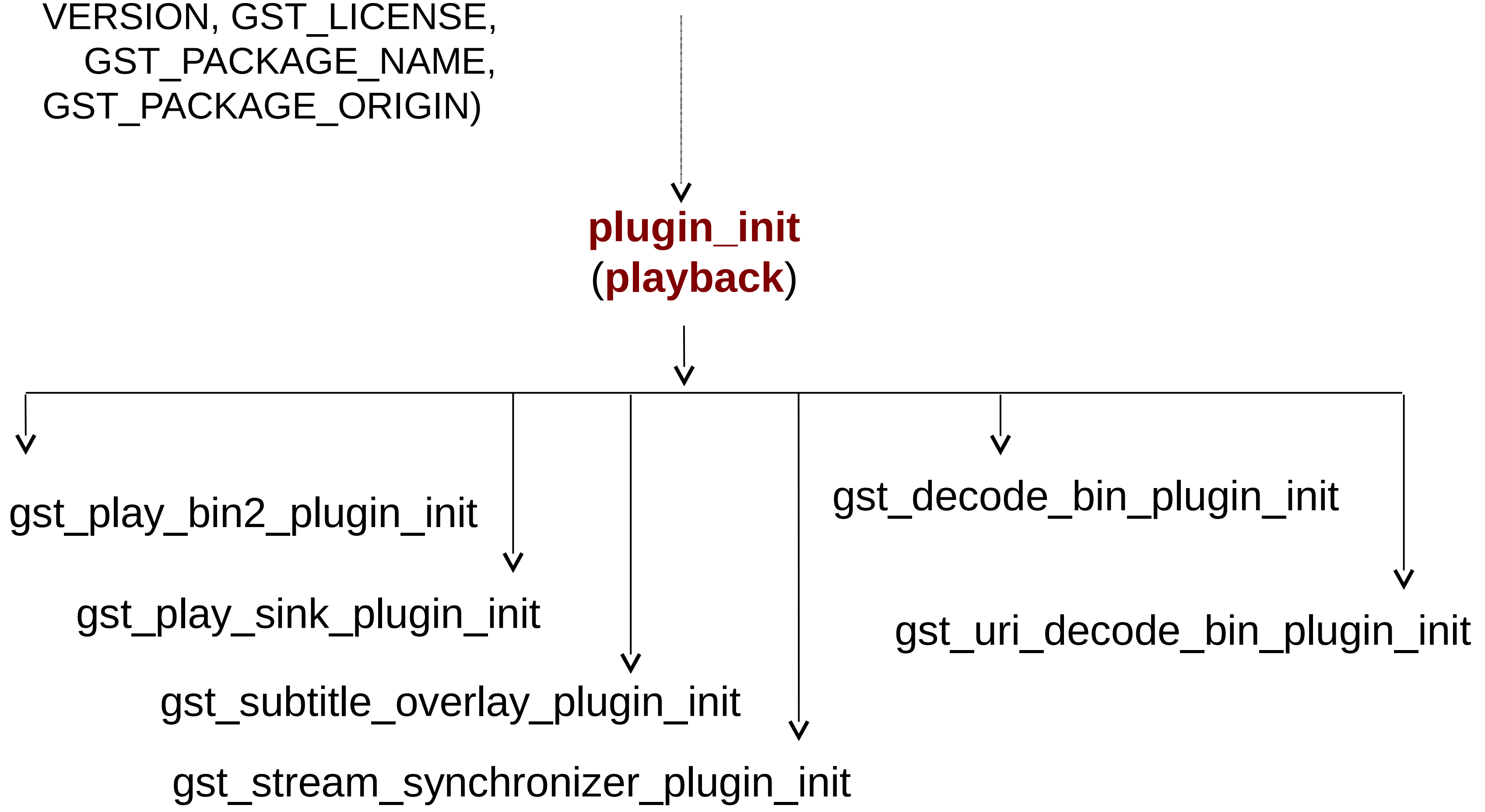
(desc->plugin_init) (plugin)

$\downarrow$

plugin_init(**playback**)
/* now go into
gst-plugins-base

gstplayback.c

```
GST_PLUGIN_DEFINE (GST_VERSION_MAJOR,
    GST_VERSION_MINOR,
    "playback",
    "various playback elements", plugin_init,
VERSION, GST_LICENSE,
    GST_PACKAGE_NAME,
GST_PACKAGE_ORIGIN)
```


plugin_init
(playback)
gst_play_bin2_plugin_init
gst_play_sink_plugin_init
gst_subtitle_overlay_plugin_init
gst_stream_synchronizer_plugin_init
gst_decode_bin_plugin_init
gst_uri_decode_bin_plugin_init

gstplaybin2.c

```
        gst_play_bin2_plugin_init
                    ↓
gst_element_register (plugin, "playbin",
GST_RANK_NONE,
     GST_TYPE_PLAY_BIN);
                    ↓
    static const GTypeInfo gst_play_bin_info = {
 ...
      (GClassInitFunc) gst_play_bin_class_init,
 ...
      (GInstanceInitFunc) gst_play_bin_init,
 ...
    };
```

g_object_new("playbin")

/* create an instance of the element

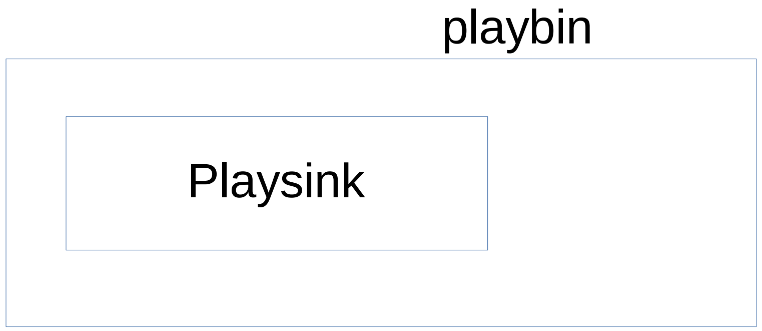
gst_element_base_class_init

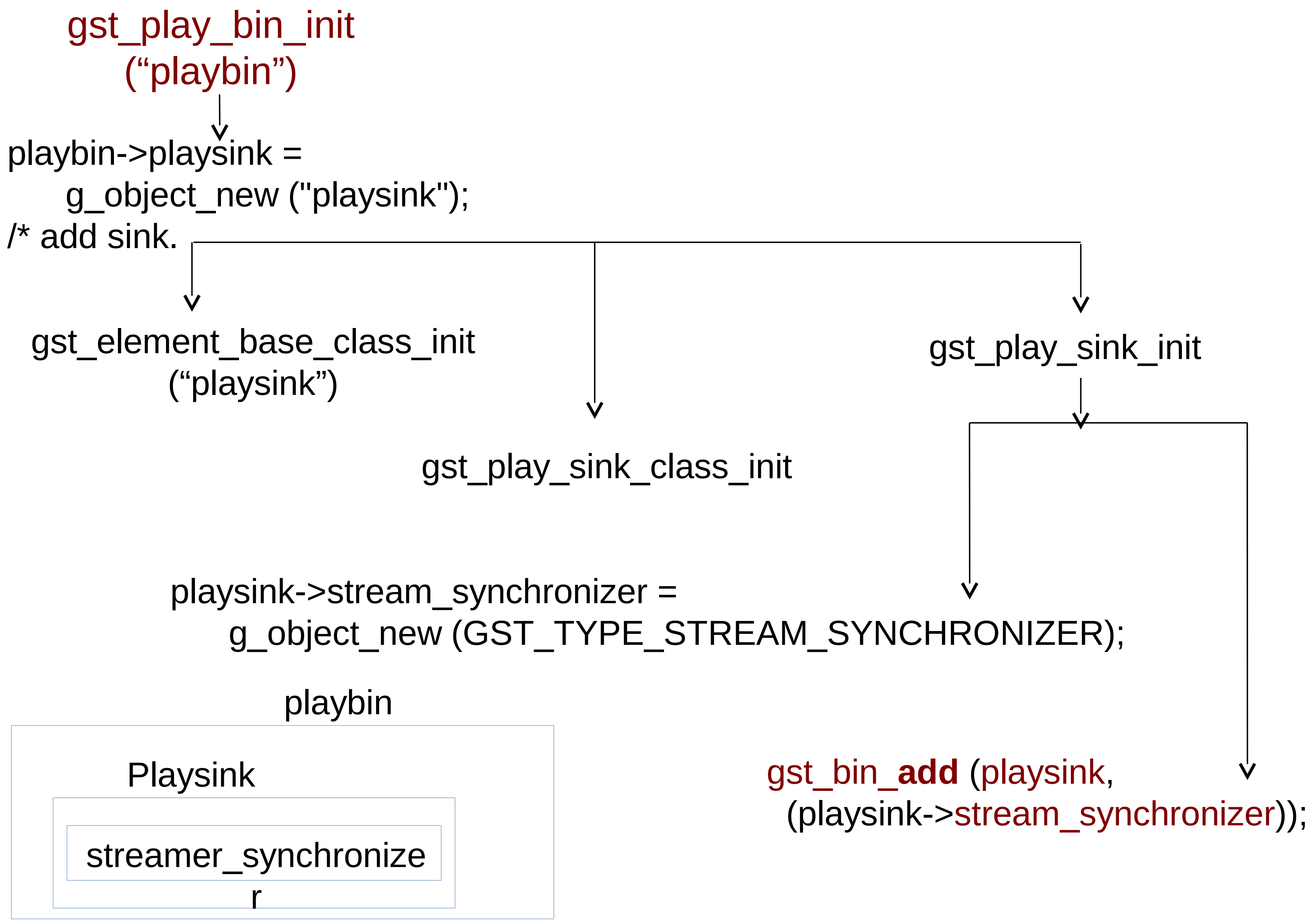
gst_play_bin_class_init

```
gobject_klass->set_property
    = gst_play_bin_set_property;
    …
gstelement_klass->change_state =
    (gst_play_bin_change_state);
    ...
```

gst_play_bin_init

```
/* add sink */
playbin->playsink =
    g_object_new ( "playsink");
gst_bin_add (playbin,playbin->playsink);
```





gst_play_bin_init
(“playbin”)

```
playbin->playsink =
    g_object_new ("playsink");
/* add sink.
```

gst_element_base_class_init
(“playsink”)

gst_play_sink_class_init

gst_play_sink_init

```
playsink->stream_synchronizer =
    g_object_new (GST_TYPE_STREAM_SYNCHRONIZER);
```

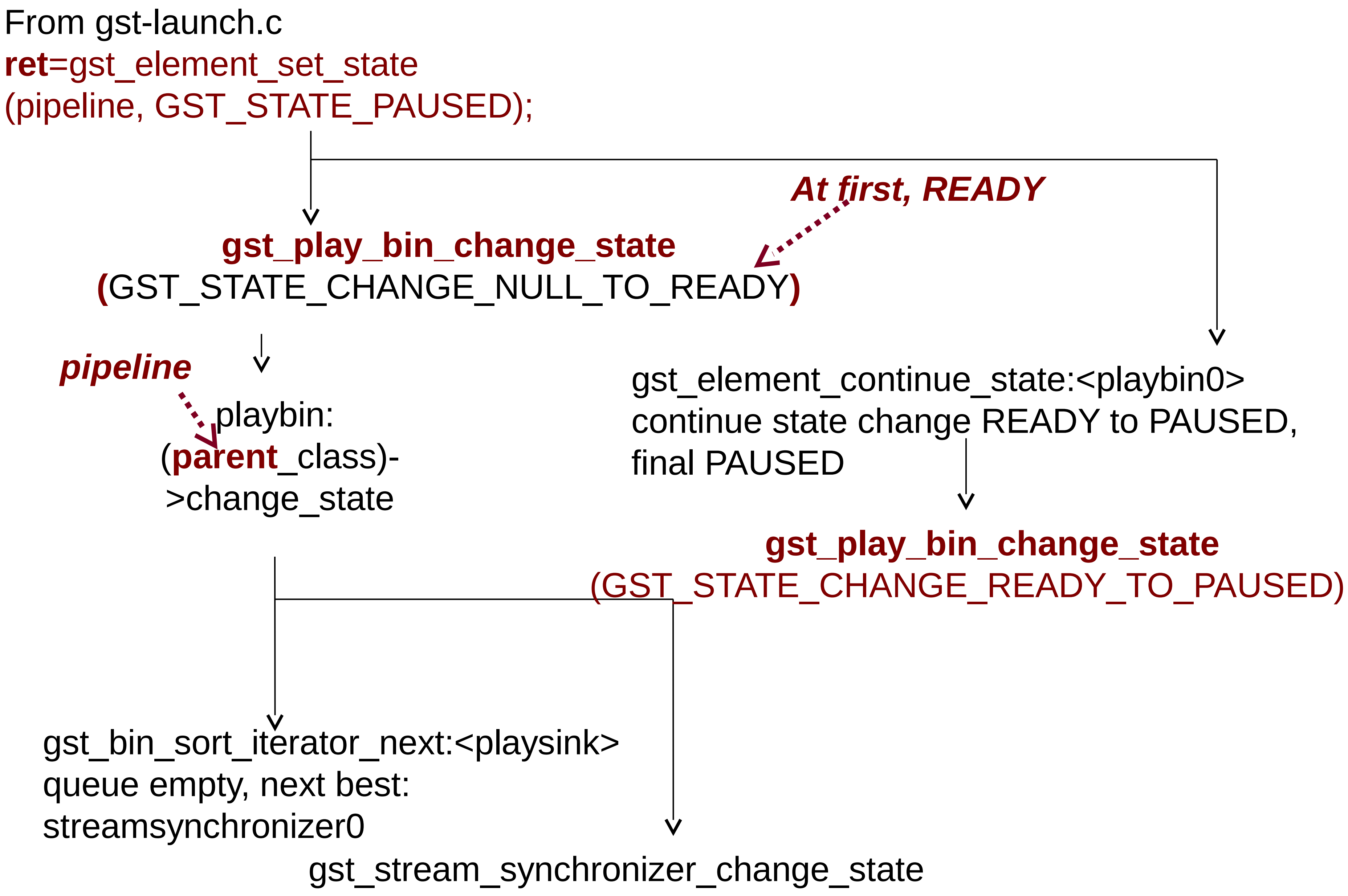
gst_bin_**add** (playsink,
(playsink->stream_synchronizer));

3, App changes state from NULL to PAUSED.

At gst-launch.c

$\downarrow$

```
ret = gst_element_set_state(pipeline,GST_STATE_PAUSED);
```



From gst-launch.c

ret=gst_element_set_state
(pipeline, GST_STATE_PAUSED);

At first, READY

gst_play_bin_change_state
(GST_STATE_CHANGE_NULL_TO_READY**)**

pipeline

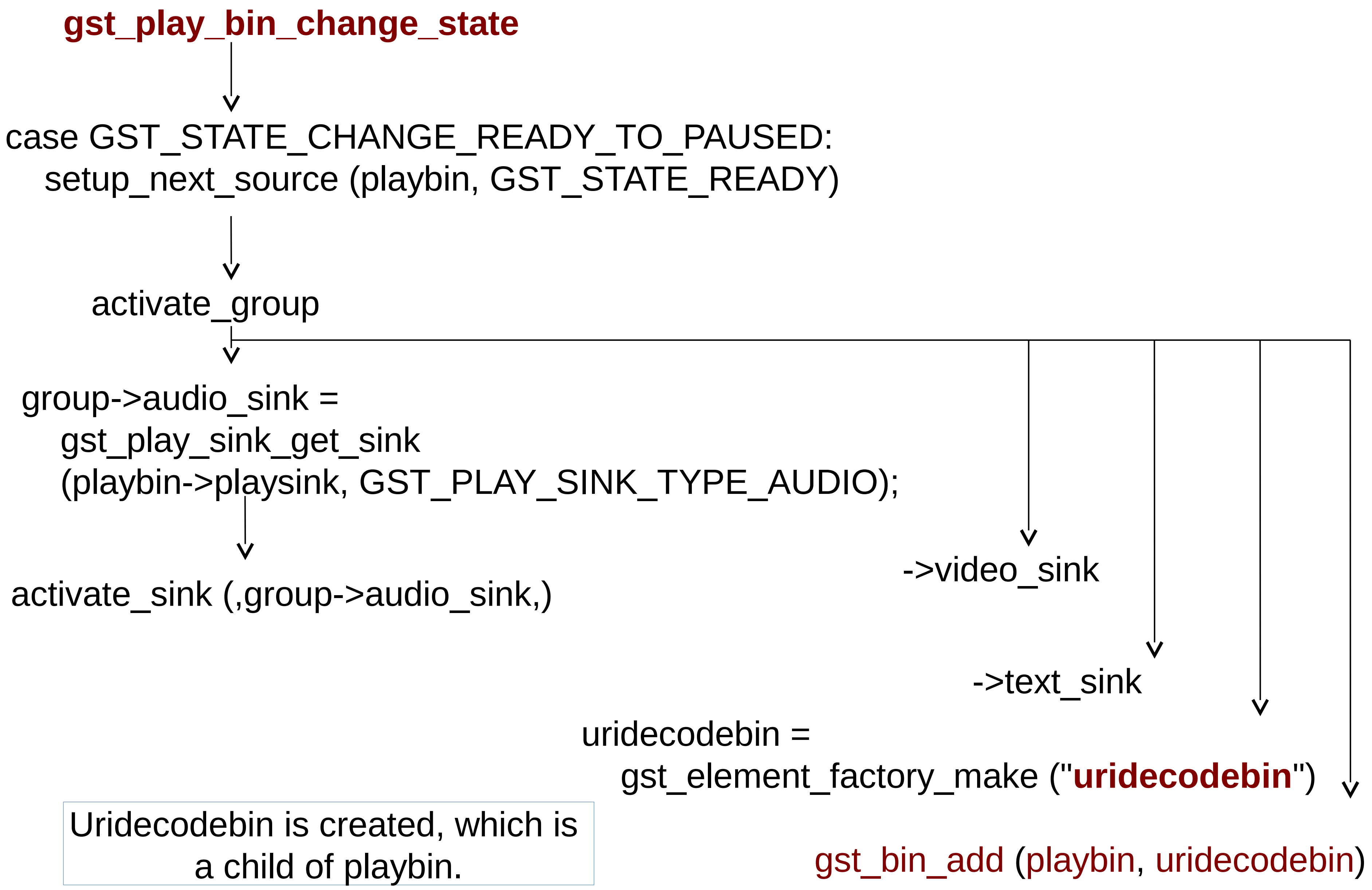
playbin:
(**parent**_class)-
>change_state

gst_element_continue_state:<playbin0>
continue state change READY to PAUSED,
final PAUSED

gst_play_bin_change_state
(GST_STATE_CHANGE_READY_TO_PAUSED)

gst_bin_sort_iterator_next:<playsink>
queue empty, next best:
streamsynchronizer0

gst_stream_synchronizer_change_state



gst_play_bin_change_state

```
case GST_STATE_CHANGE_READY_TO_PAUSED:
    setup_next_source (playbin, GST_STATE_READY)
```

```
activate_group
```

```
group->audio_sink =
    gst_play_sink_get_sink
    (playbin->playsink, GST_PLAY_SINK_TYPE_AUDIO);
```

```
activate_sink (,group->audio_sink,)
```

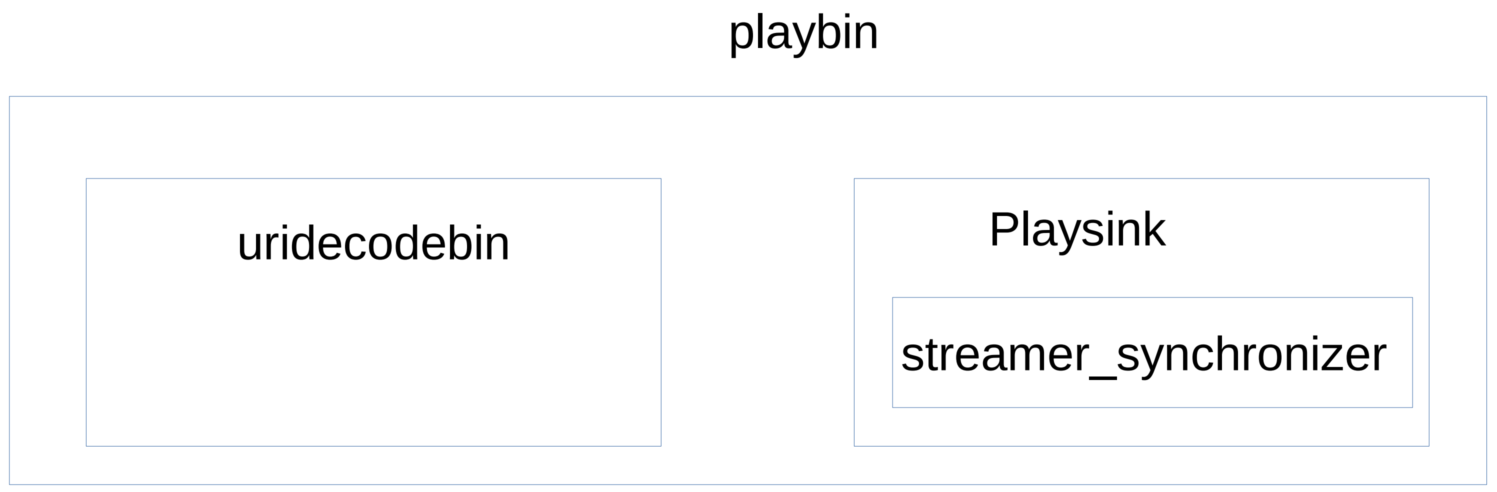
```
->video_sink
```

```
->text_sink
```

```
uridecodebin =
    gst_element_factory_make ("uridecodebin")
```

```
gst_bin_add (playbin, uridecodebin)
```

Uridecodebin is created, which is a child of playbin.


playbin
uridecodebin
Playsink
streamer_synchronizer

gst_play_bin_change_state
GST_STATE_CHANGE_READY_TO_PAUSED

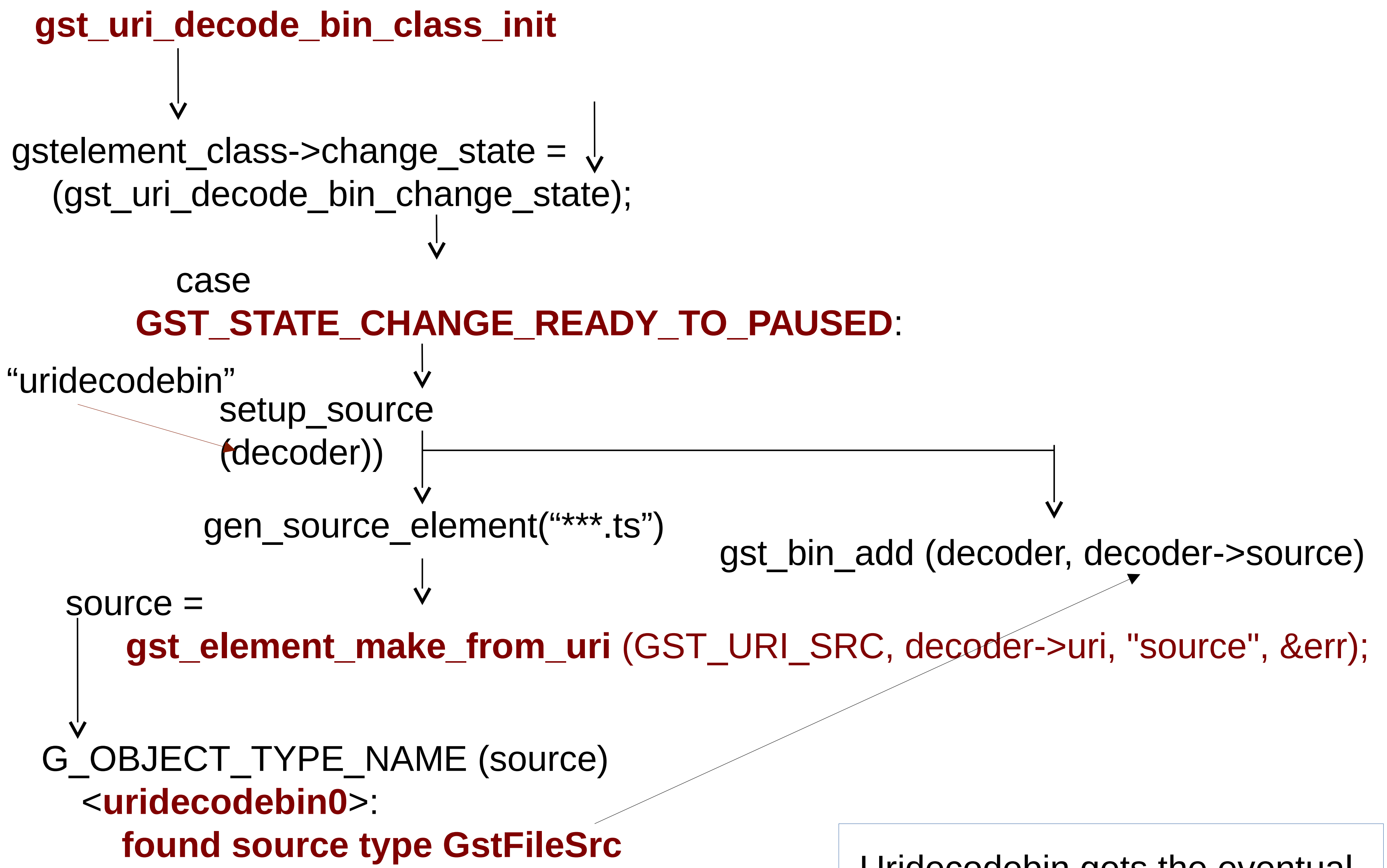
(parent_class)->change_state

gst_stream_synchronizer_change_state

gst_play_sink_change_state

/* we want to go async to PAUSED
until we managed to configure and add the
* sinks */
do_async_start

(gst_play_sink_parent_class)->handle_message
(playsink, message)



gst_uri_decode_bin_class_init

gstelement_class->change_state =
(gst_uri_decode_bin_change_state);

case
GST_STATE_CHANGE_READY_TO_PAUSED:

“uridecodebin”

setup_source
(decoder))

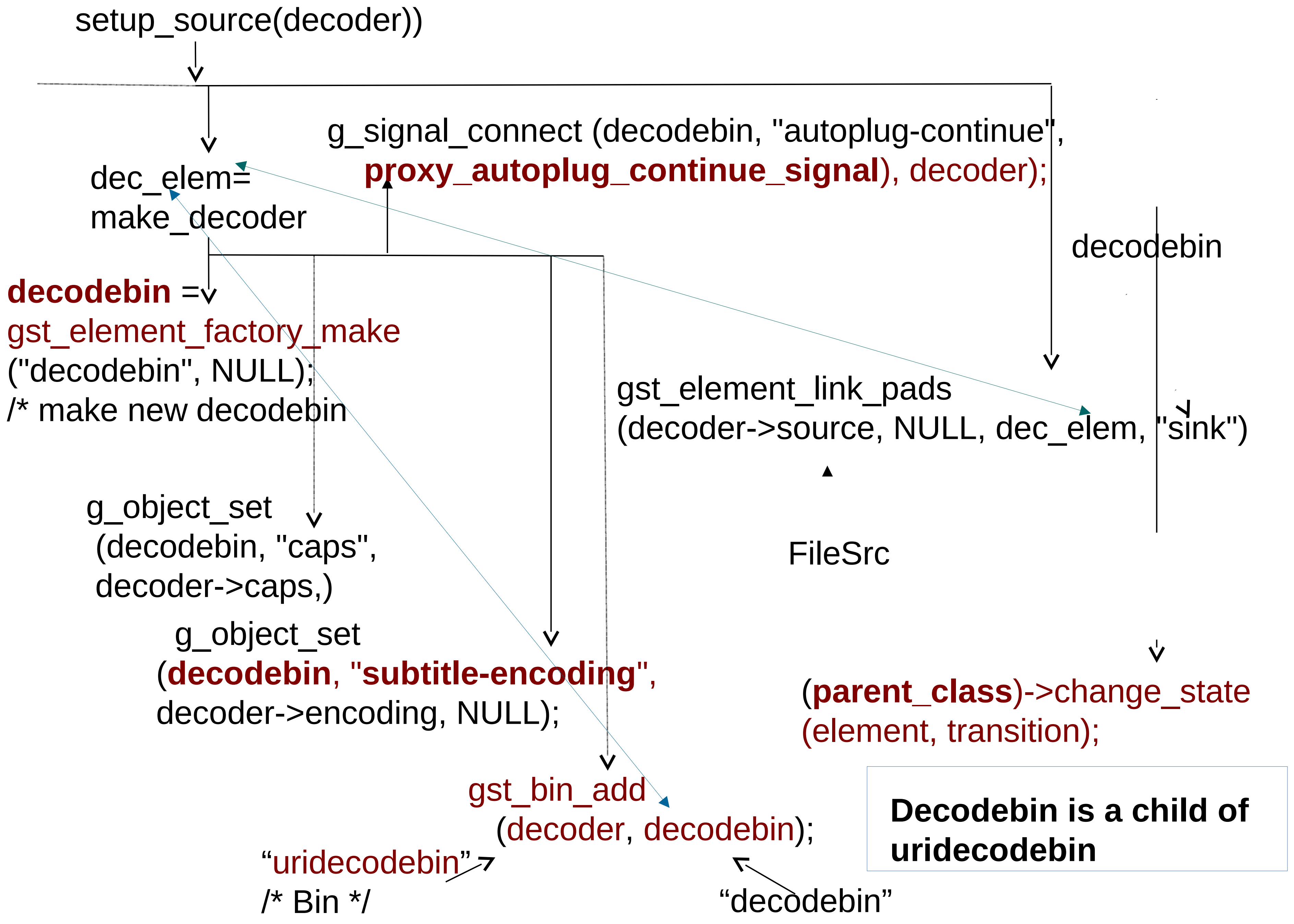
gen_source_element(“***.ts”)

gst_bin_add (decoder, decoder->source)

source =
gst_element_make_from_uri (GST_URI_SRC, decoder->uri, "source", &err);

G_OBJECT_TYPE_NAME (source)
<**uridecodebin0**>:
found source type GstFileSrc

Uridecodebin gets the eventual source, which is FileSrc



setup_source(decoder))

dec_elem= make_decoder

g_signal_connect (decodebin, "autoplug-continue", **proxy_autoplug_continue_signal**), decoder);

decodebin

decodebin = gst_element_factory_make ("decodebin", NULL); /* make new decodebin

gst_element_link_pads (decoder->source, NULL, dec_elem, "sink")

g_object_set (decodebin, "caps", decoder->caps,)

FileSrc

g_object_set (**decodebin**, "**subtitle-encoding**", decoder->encoding, NULL);

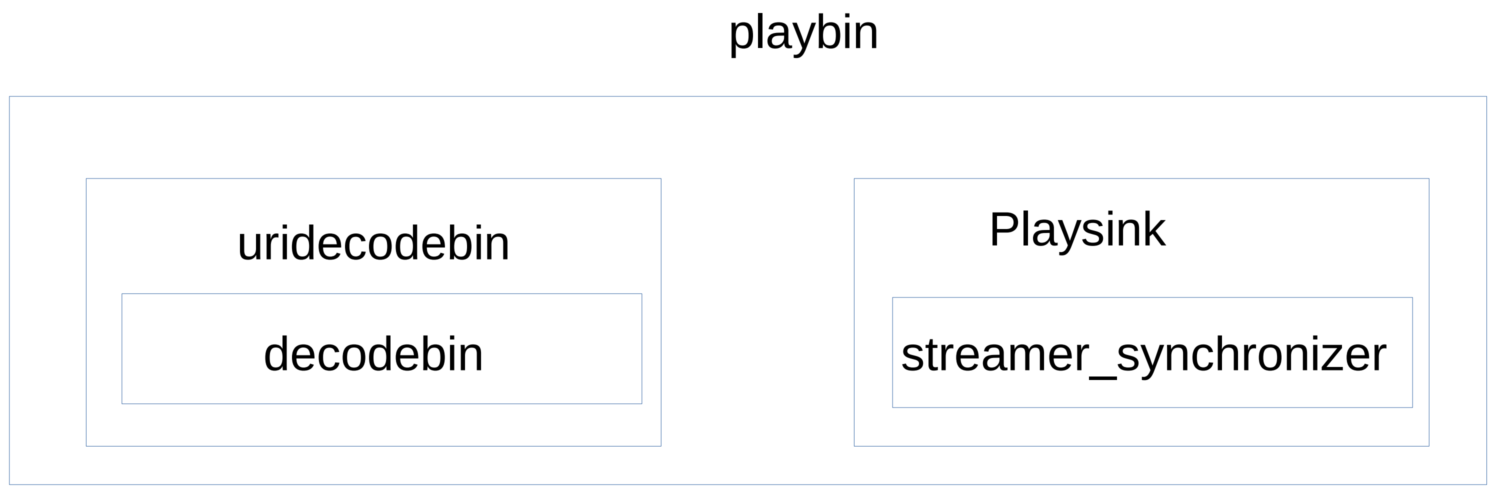
(**parent_class**)->change_state (element, transition);

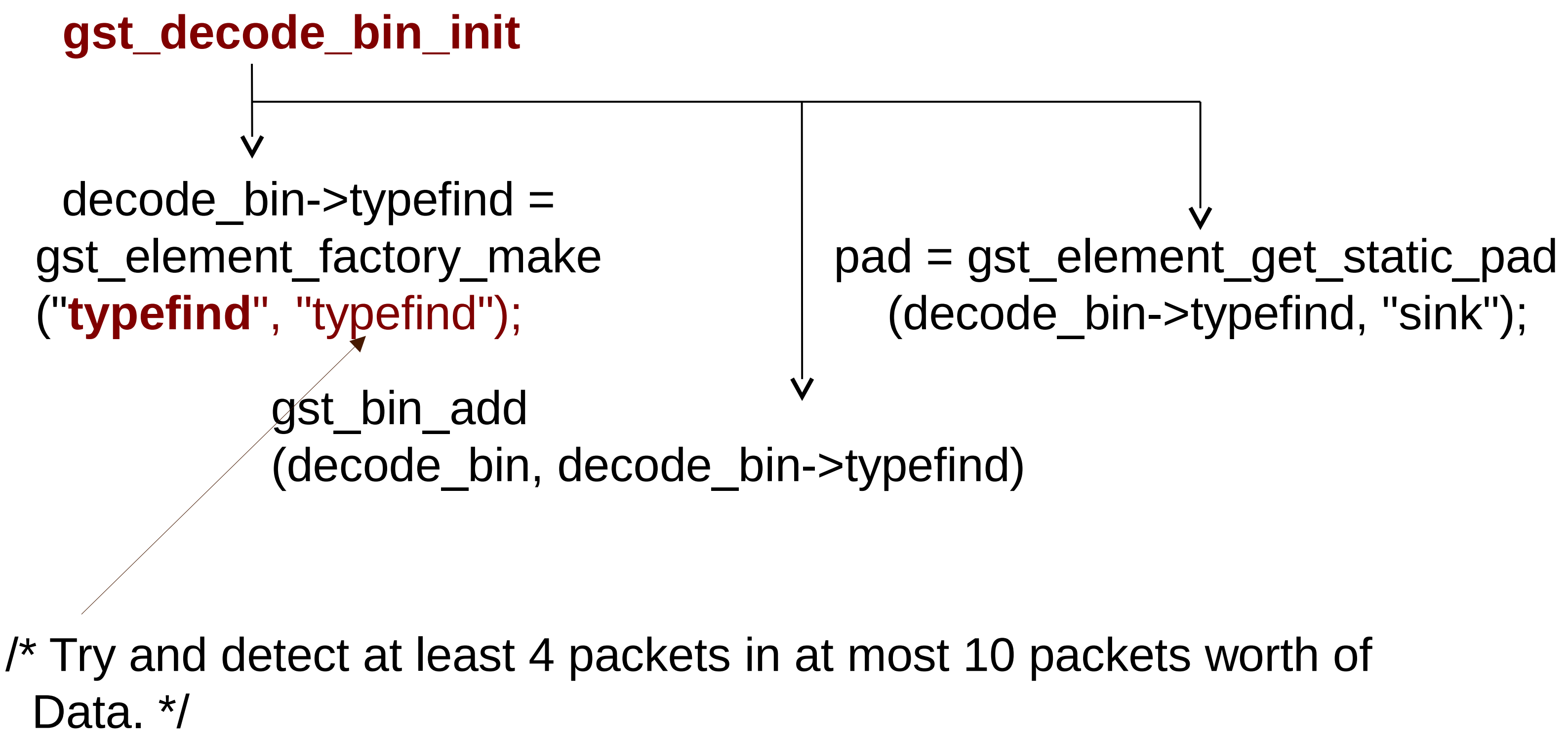
gst_bin_add (decoder, decodebin);

"uridecodebin" /* Bin */

"decodebin"

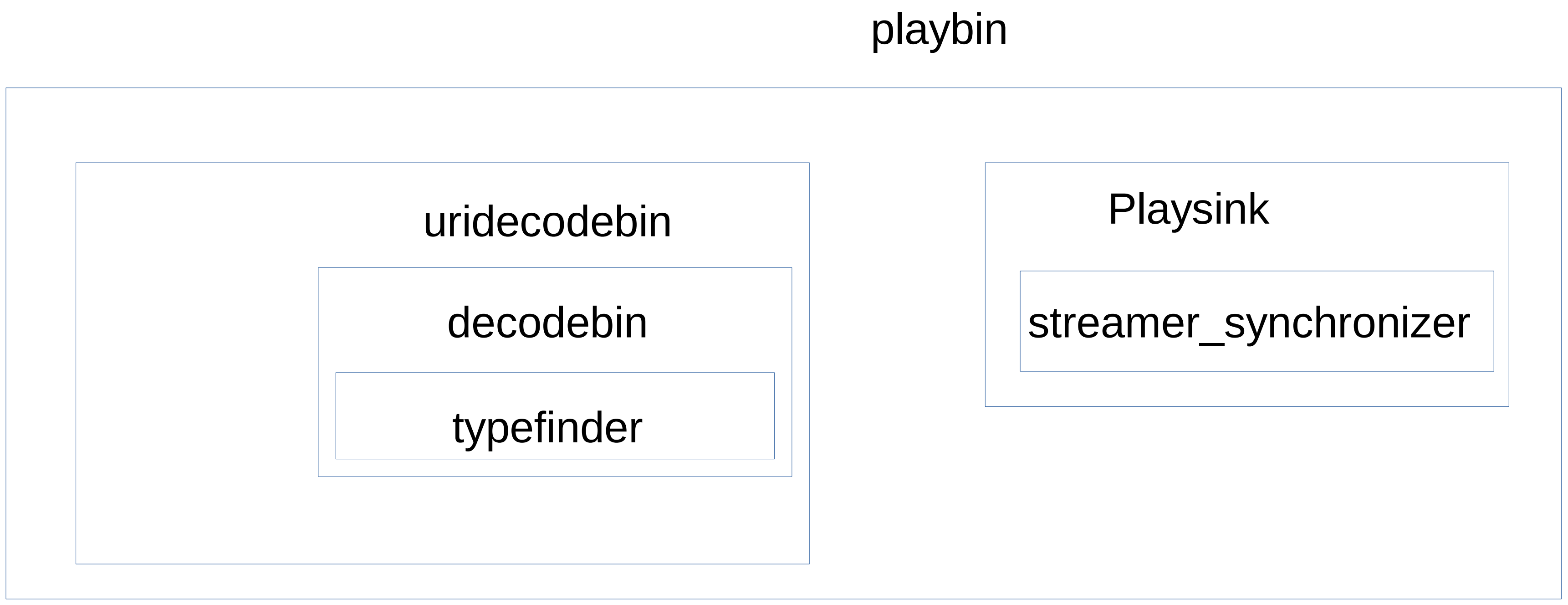
Decodebin is a child of uridecodebin


playbin
uridecodebin
decodebin
Playsink
streamer_synchronizer



```
gst_decode_bin_init

decode_bin->typefind =
gst_element_factory_make
("typefind", "typefind");

gst_bin_add
(decode_bin, decode_bin->typefind)

pad = gst_element_get_static_pad
    (decode_bin->typefind, "sink");

/* Try and detect at least 4 packets in at most 10 packets worth of
  Data. */
```

Typefinder is a child of decodebin.


playbin
uridecodebin
decodebin
typefinder
Playsink
streamer_synchronizer

gst_decode_bin_change_state
GST_STATE_CHANGE_NULL_TO_READY

gst_decode_bin_change_state
GST_STATE_CHANGE_READY_TO_PAUSED

```
/* connect a signal to find out when the typefind element found
 * a type */
dbin->have_type_id =
    g_signal_connect (dbin->typefind, "have-type",
    G_CALLBACK (type_found), dbin);
```

4, App enters event_loop

At gst-launch.c

↓

ret =
gst_element_set_state(pipeline,GST_STATE_PAUSED)

↓

GST_STATE_CHANGE_ASYNC

↓

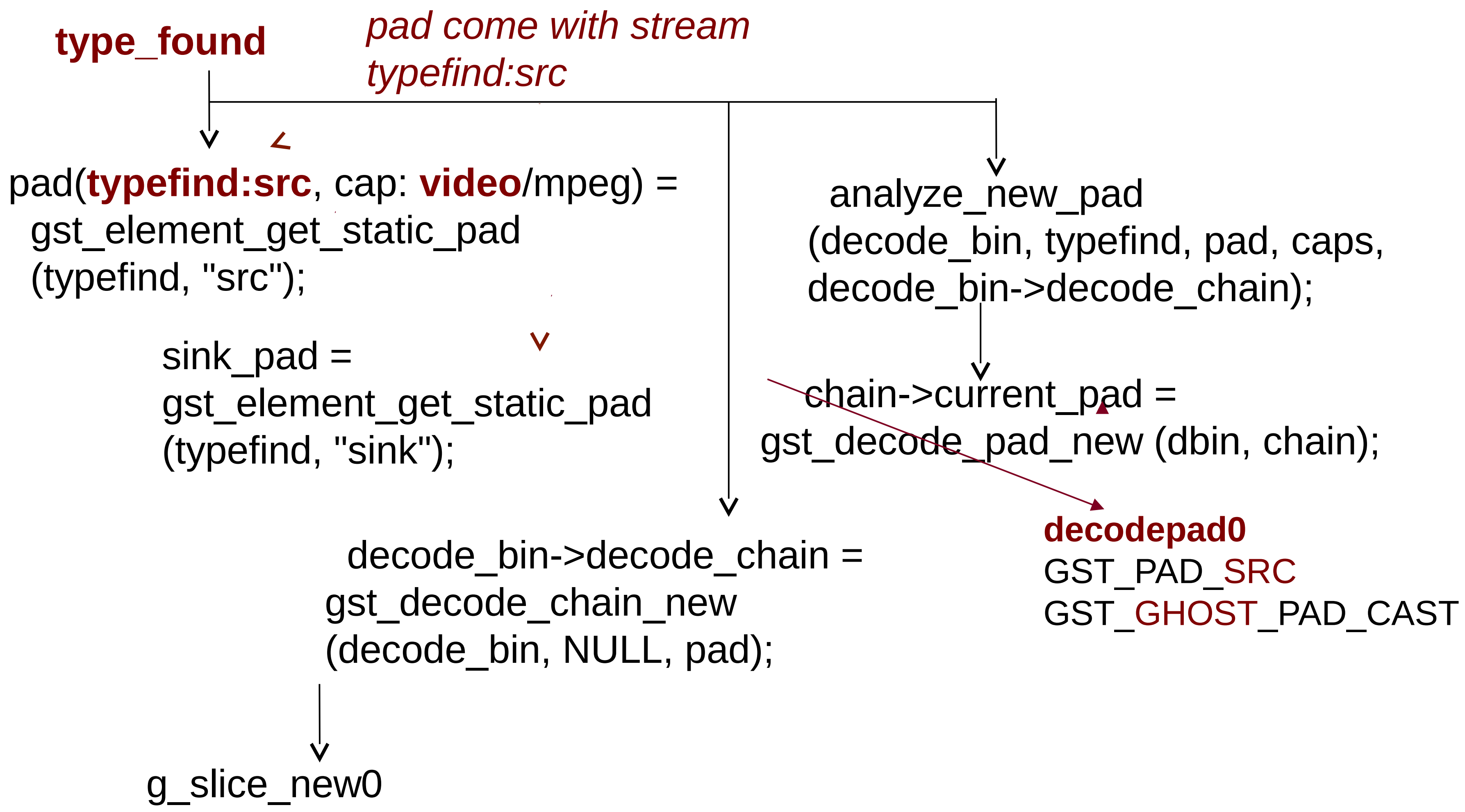
event_loop

type_found

typefind found caps
video/mpegts,
packetsize=(int)188

```
GST_PLUGIN_DEFINE (
    typefindfunctions,
    "default typefind functions",
    plugin_init, VERSION, GST_LICENSE,
GST_PACKAGE_NAME, GST_PACKAGE_ORIGIN)
```

```
  TYPE_FIND_REGISTER
(plugin, "video/mpegts",
     mpeg_ts_type_find, "ts,mts", MPEGTS_CAPS);
```



type_found

pad come with stream typefind:src

pad(**typefind:src**, cap: **video**/mpeg) =
gst_element_get_static_pad
(typefind, "src");

sink_pad =
gst_element_get_static_pad
(typefind, "sink");

decode_bin->decode_chain =
gst_decode_chain_new
(decode_bin, NULL, pad);

g_slice_new0

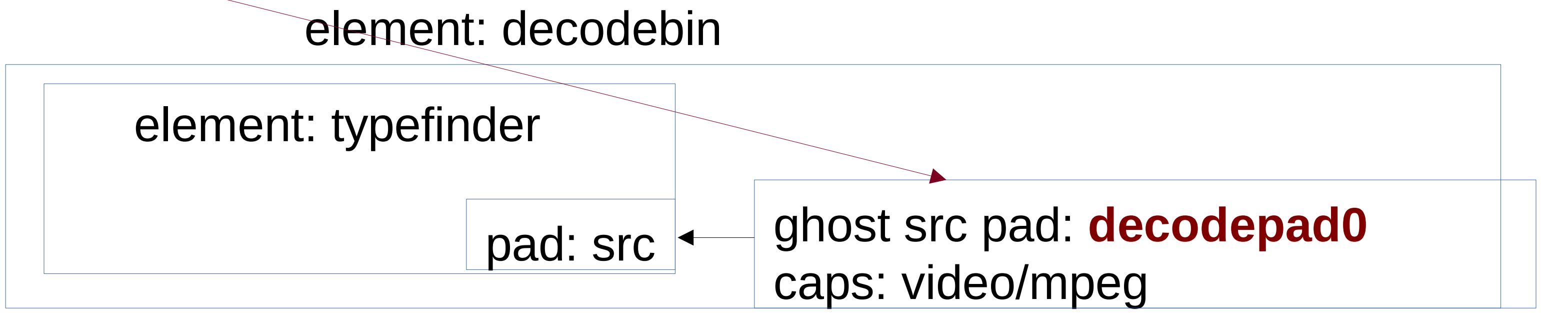
analyze_new_pad
(decode_bin, typefind, pad, caps,
decode_bin->decode_chain);

chain->current_pad =
gst_decode_pad_new (dbin, chain);

decodepad0
GST_PAD_SRC
GST_GHOST_PAD_CAST

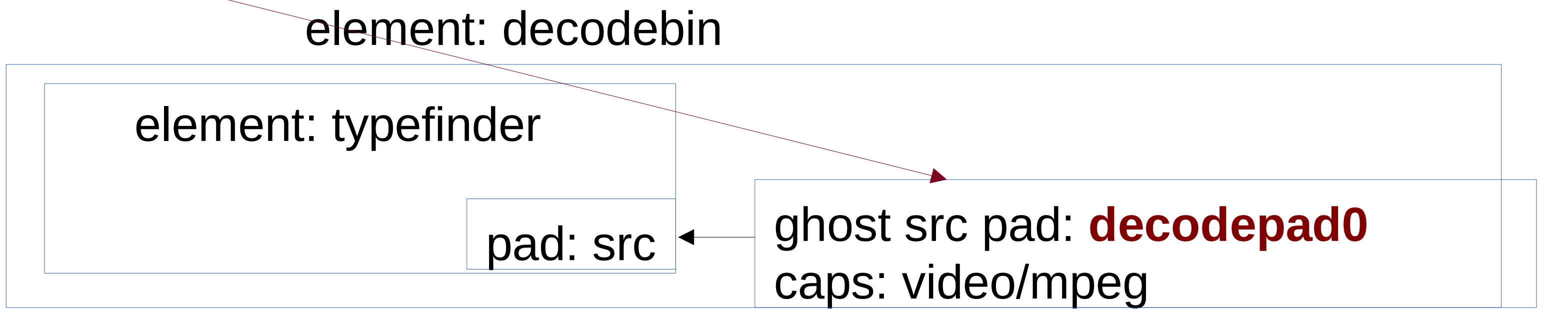
analyze_new_pad

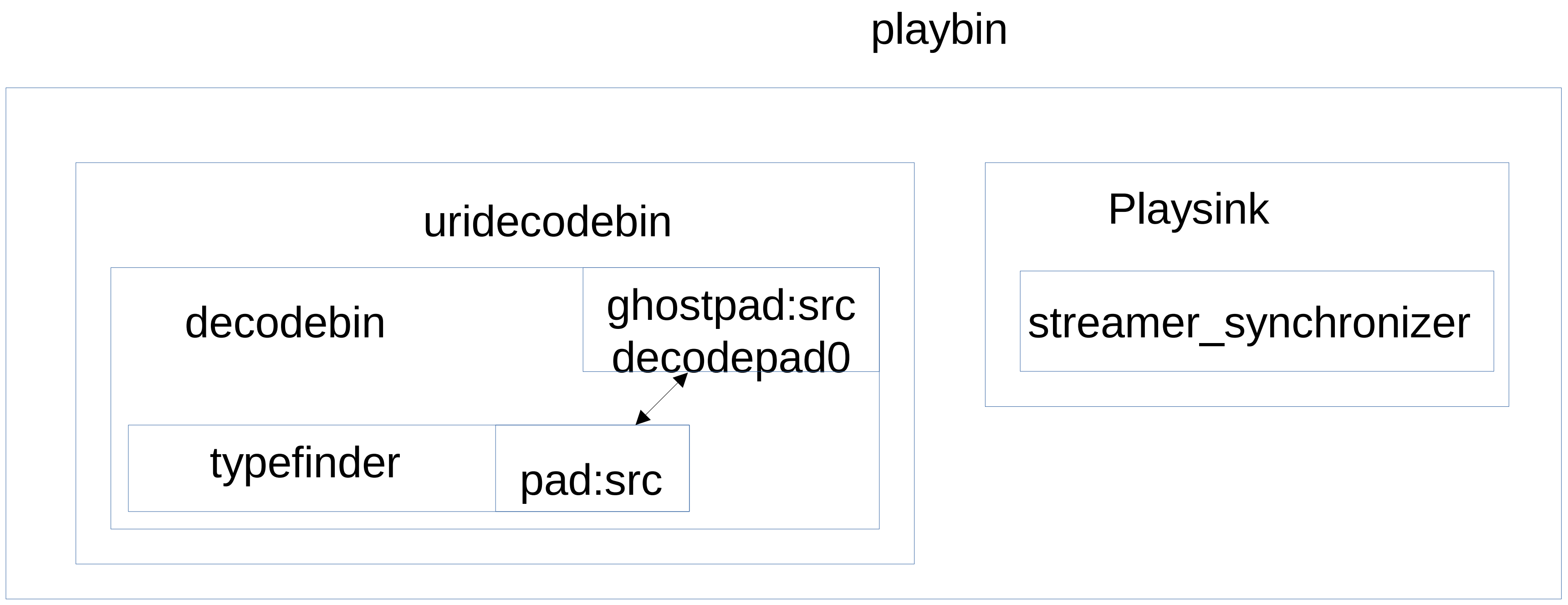
```
chain->current_pad = gst_decode_pad_new (dbin, chain)
   /* Creates a new GstDecodePad for the given pad. */

pad_tmpl = gst_static_pad_template_get (&decoder_bin_src_template);
  dpad =
   g_object_new (GST_TYPE_DECODE_PAD, "direction", GST_PAD_SRC,
     "template", pad_tmpl,);
  gst_ghost_pad_construct (GST_GHOST_PAD_CAST (dpad));

   ppad = gst_proxy_pad_get_internal (GST_PROXY_PAD (dpad));
   gst_pad_set_query_function (GST_PAD_CAST (ppad), gst_decode_pad_query);

     return dpad;
```



analyze_new_pad

```
chain->current_pad = gst_decode_pad_new (dbin, chain)
   /* Creates a new GstDecodePad for the given pad. */

pad_tmpl = gst_static_pad_template_get (&decoder_bin_src_template);
  dpad =
   g_object_new (GST_TYPE_DECODE_PAD, "direction", GST_PAD_SRC,
     "template", pad_tmpl,);
  gst_ghost_pad_construct (GST_GHOST_PAD_CAST (dpad));

   ppad = gst_proxy_pad_get_internal (GST_PROXY_PAD (dpad));
   gst_pad_set_query_function (GST_PAD_CAST (ppad), gst_decode_pad_query);

     return dpad;
```




playbin
uridecodebin
decodebin
ghostpad:src
decodepad0
typefinder
pad:src
Playsink
streamer_synchronizer

analyze_new_pad

```
g_signal_emit (dbin,
gst_decode_bin_signals[SIGNAL_AUTOPLUG_CONTINUE],
, dpad, );
```

```
gst_decode_bin_signals[SIGNAL_AUTOPLUG_CONTINUE] =
  g_signal_new ("autoplug-continue",G_SIGNAL_RUN_LAST,autoplug_continue),
  _gst_boolean_accumulator );

          activate_group
          /* playback */

group->autoplug_continue_id =
    g_signal_connect (uridecodebin, "autoplug-continue",
    G_CALLBACK (autoplug_continue_cb), group);

At uridecodebin,
   g_signal_connect (decodebin, "autoplug-continue",
      G_CALLBACK (proxy_autoplug_continue_signal), decoder);

 2, g_signal_emit (dec,
    gst_uri_decode_bin_signals[SIGNAL_AUTOPLUG_CONTINUE], 0, pad, caps,
    &result);


 1, g_signal_emit (G_OBJECT (dbin),
   gst_decode_bin_signals[SIGNAL_AUTOPLUG_CONTINUE], 0, dpad, caps,
   &apcontinue);
```

analyze_new_pad

```
/* get the factories */
g_signal_emit (dbin,
  gst_decode_bin_signals
[SIGNAL_AUTOPLUG_FACTORIES,);
```

autoplug_factories_cb

/* Called when we must provide
a list of factories to plug to
@pad with @caps.

```
gst_play_bin_update_elements_list
 (playbin)
```

```
res = gst_element_factory_list_get_elements
  (GST_ELEMENT_FACTORY_TYPE_DECODABLE,);
tmp = gst_element_factory_list_get_elements
  (GST_ELEMENT_FACTORY_TYPE_AUDIOVIDEO_SINKS,);
   playbin->elements = g_list_concat (res, tmp);
```

/* The caps should be set when a
Plugin was created.*/

/* Filter out all the elementfactories
in @**list** that can handle @**caps** in
the given direction.*/

```
factory_list =
gst_element_factory_list_filter
(playbin->elements, caps,
GST_PAD_SINK,);
```

autoplug_continue_cb
/* decodebin */

↓

gst_subtitle_overlay_create_factory_caps
/* subtibleoverlaybin */

↓

factories = gst_registry_feature_filter (registry,
_factory_filter, **FALSE**, &_factory_caps);

↓

/* Only retrieve the first if TRUE */

_factory_filter
/* Get all features of **subtitles**
other than creating factory */

↓

/* Done with the first signal
handler for autoplug_continue

analyze_new_pad

```
g_signal_emit (dbin,
  gst_decode_bin_signals
  [SIGNAL_AUTOPLUG_SORT],
  0, dpad, caps, factories, &result);
```

analyze_new_pad

/* Try to connect the given pad
to an element
created from one of the factories,
and recursively. */
connect_pad

g_signal_emit (dbin,
gst_decode_bin_signals
[SIGNAL_AUTOPLUG_SELECT],factory,);

autoplug_select_cb
/* We are asked to select an element.

checking factory
tsdemux

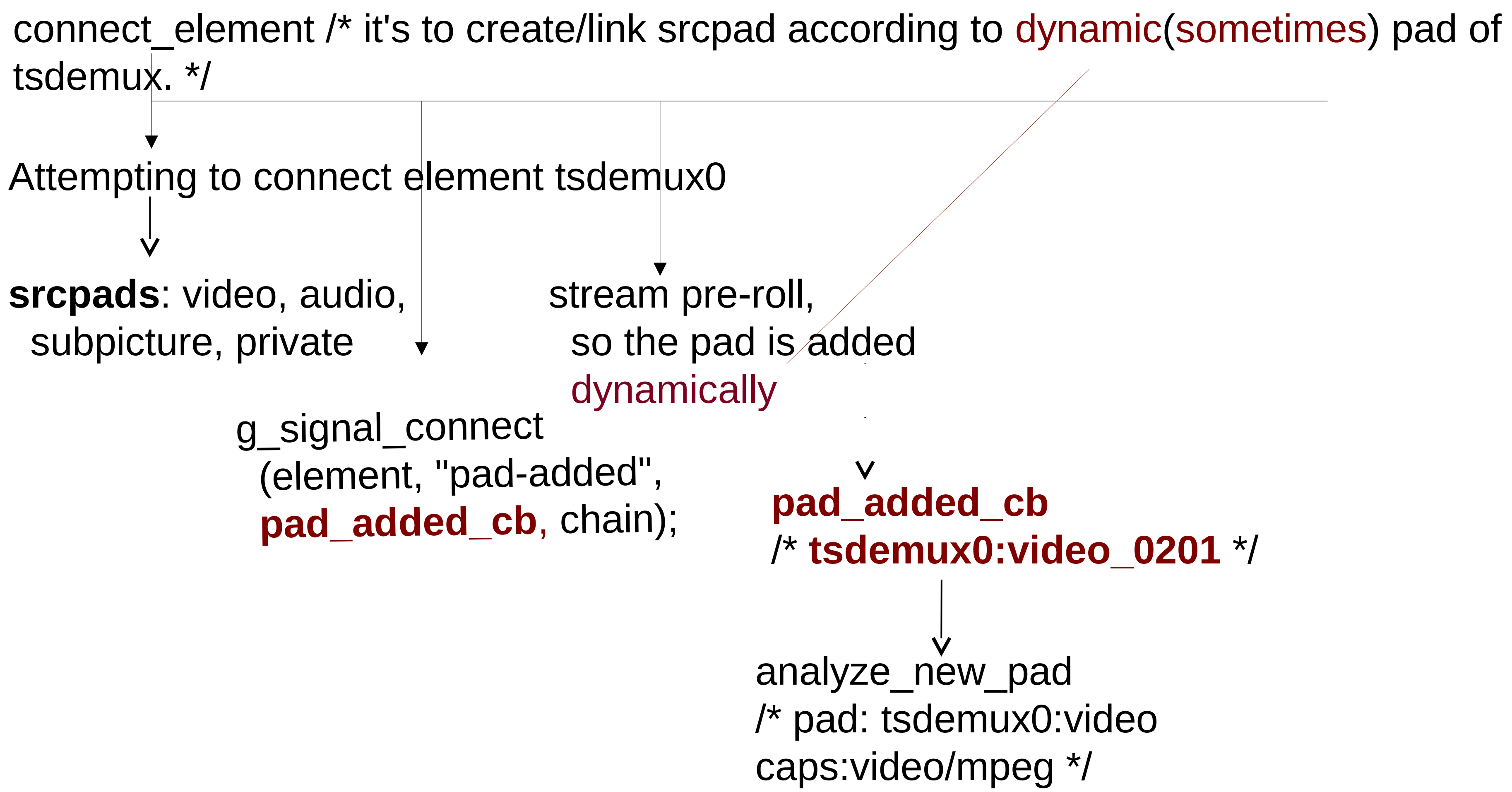
/* Try to create an element */
element =
gst_element_factory_create (factory,)

/* ... add it ... */
gst_bin_add (dbin, element)

/* ... and try to link */
gst_pad_link (pad, sinkpad)

link srcpad of typefinder to
sinkpad of tsdemux

/* link this element further */
connect_element



connect_element /* it's to create/link srcpad according to dynamic(sometimes) pad of tsdemux. */

Attempting to connect element tsdemux0

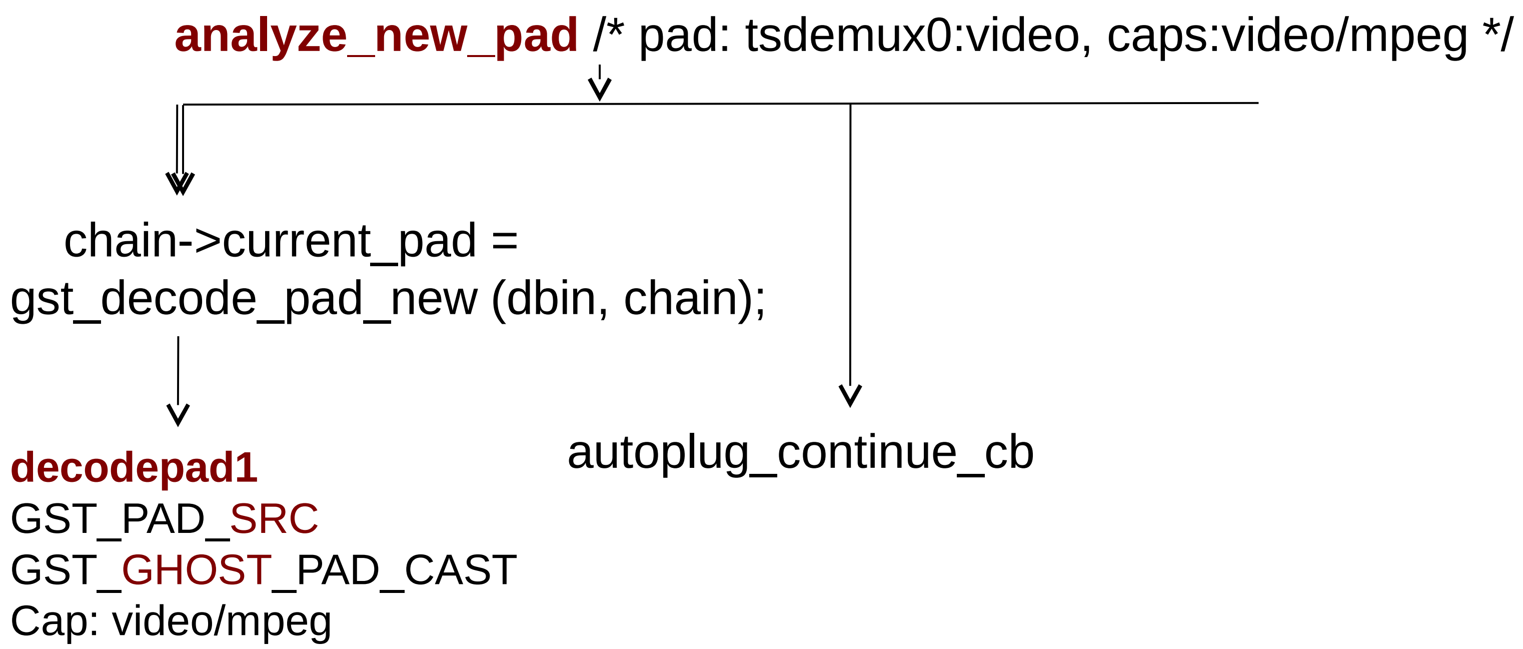
srcpads: video, audio, subpicture, private

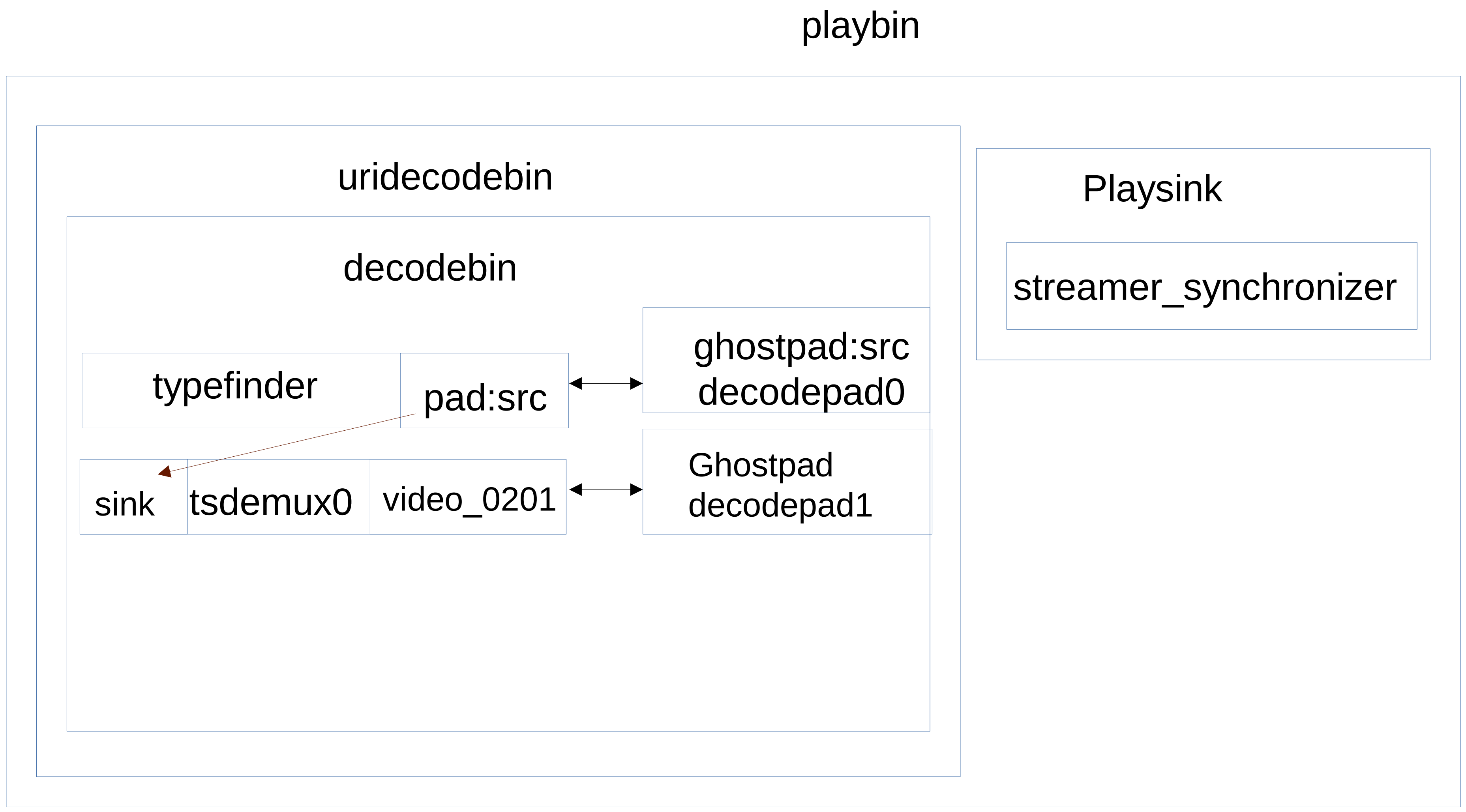
g_signal_connect (element, "pad-added", **pad_added_cb**, chain);

stream pre-roll, so the pad is added dynamically

pad_added_cb
/* **tsdemux0:video_0201** */

analyze_new_pad
/* pad: tsdemux0:video caps:video/mpeg */


analyze_new_pad /* pad: tsdemux0:video, caps:video/mpeg */
chain->current_pad = gst_decode_pad_new (dbin, chain);
autoplug_continue_cb
decodepad1
GST_PAD_SRC
GST_GHOST_PAD_CAST
Cap: video/mpeg


playbin
uridecodebin
decodebin
typefinder
pad:src
ghostpad:src
decodepad0
sink
tsdemux0
video_0201
Ghostpad
decodepad1
Playsink
streamer_synchronizer

analyze_new_pad

/* element: decodebin0; pad: tsdemux0:video; cap: video */

autoplug_factories_cb

connect_pad

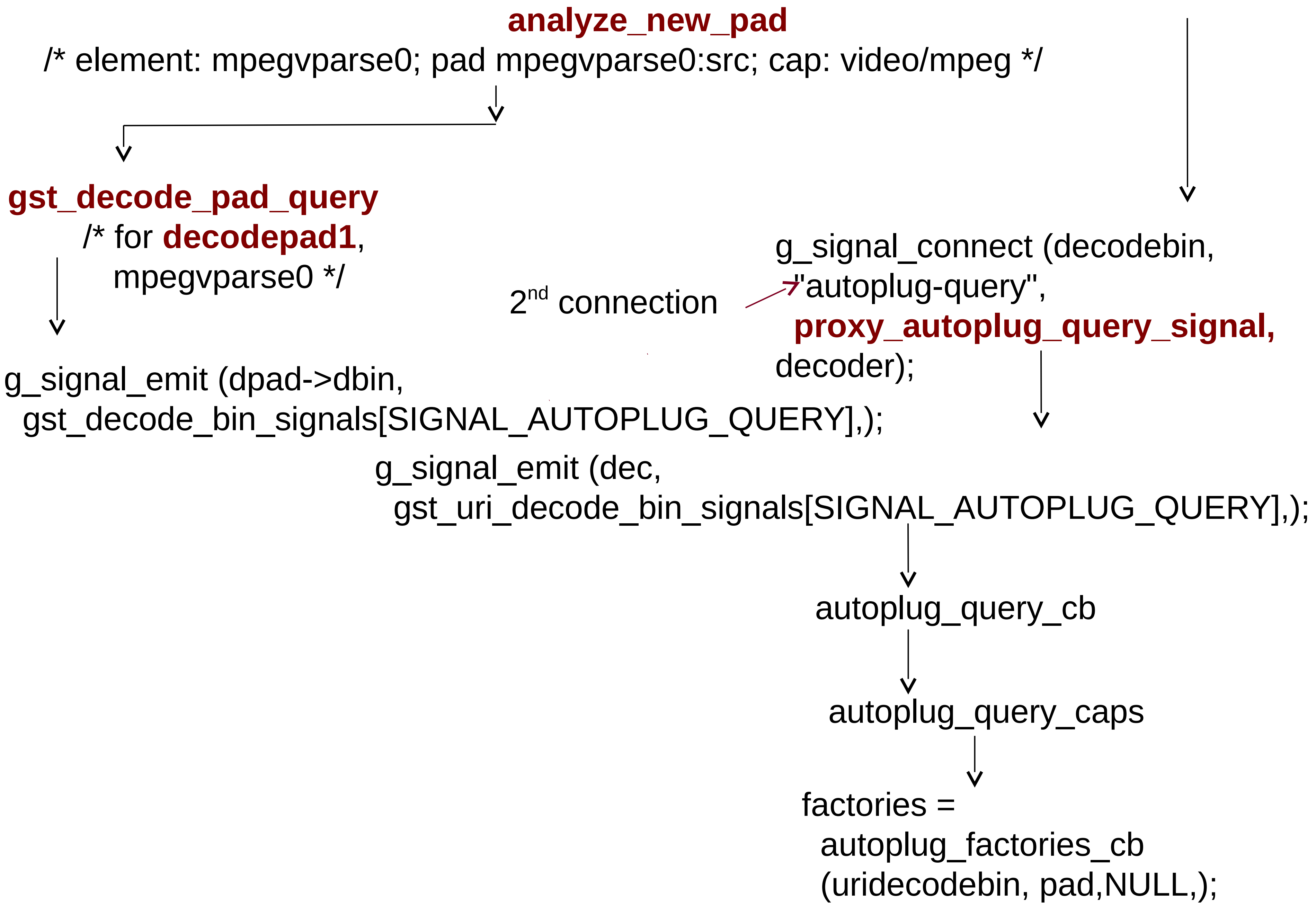
autoplug_select_cb
/* **decodepad1**,
caps: video/mpeg
checking factory **mpegvideoparse** */

gst_pad_link
/* link
srcpad of tsdemux0 and
sinkpad of mpegvparse0
*/

connect_element
/* link this element further (to mpegvparse0) */

analyze_new_pad/*
mpegvparse0 */

g_signal_emit (dpad->dbin,
gst_decode_bin_signals
[SIGNAL_AUTOPLUG_QUERY], 0);



analyze_new_pad

/* element: mpegvparse0; pad mpegvparse0:src; cap: video/mpeg */

gst_decode_pad_query

/* for **decodepad1**,
mpegvparse0 */

g_signal_emit (dpad->dbin,
gst_decode_bin_signals[SIGNAL_AUTOPLUG_QUERY],);

2nd connection

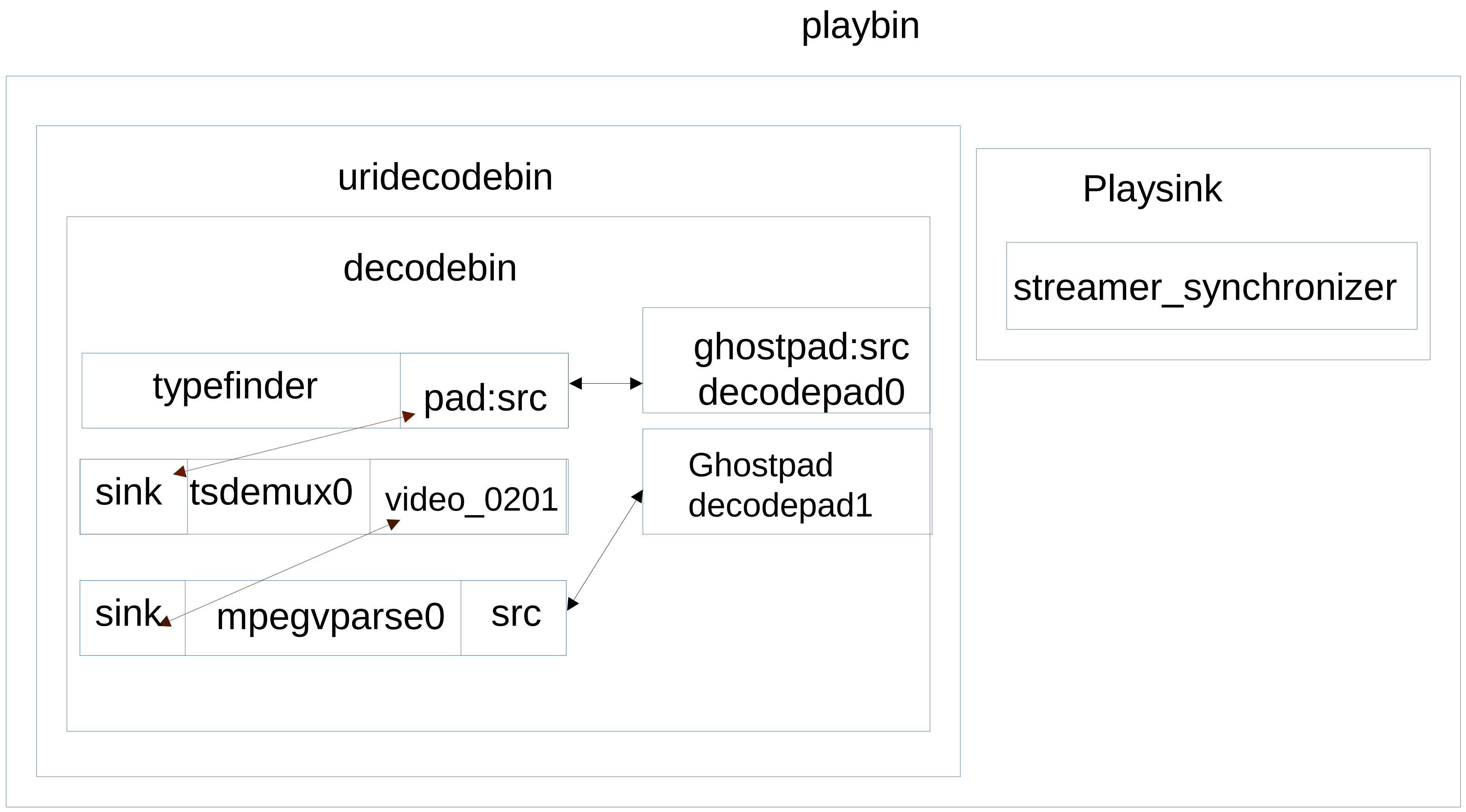
g_signal_connect (decodebin,
"autoplug-query",
proxy_autoplug_query_signal,
decoder);

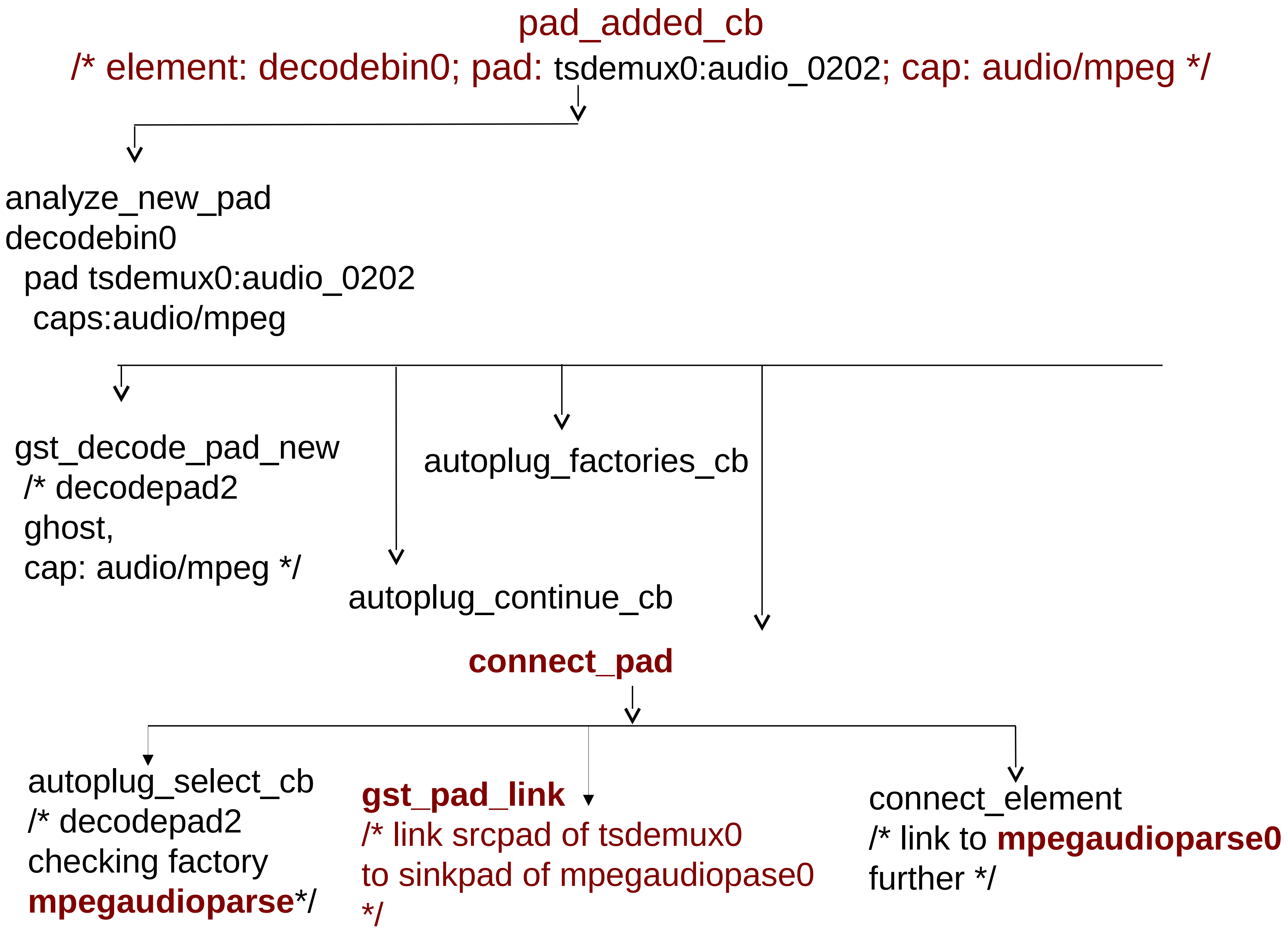
g_signal_emit (dec,
gst_uri_decode_bin_signals[SIGNAL_AUTOPLUG_QUERY],);

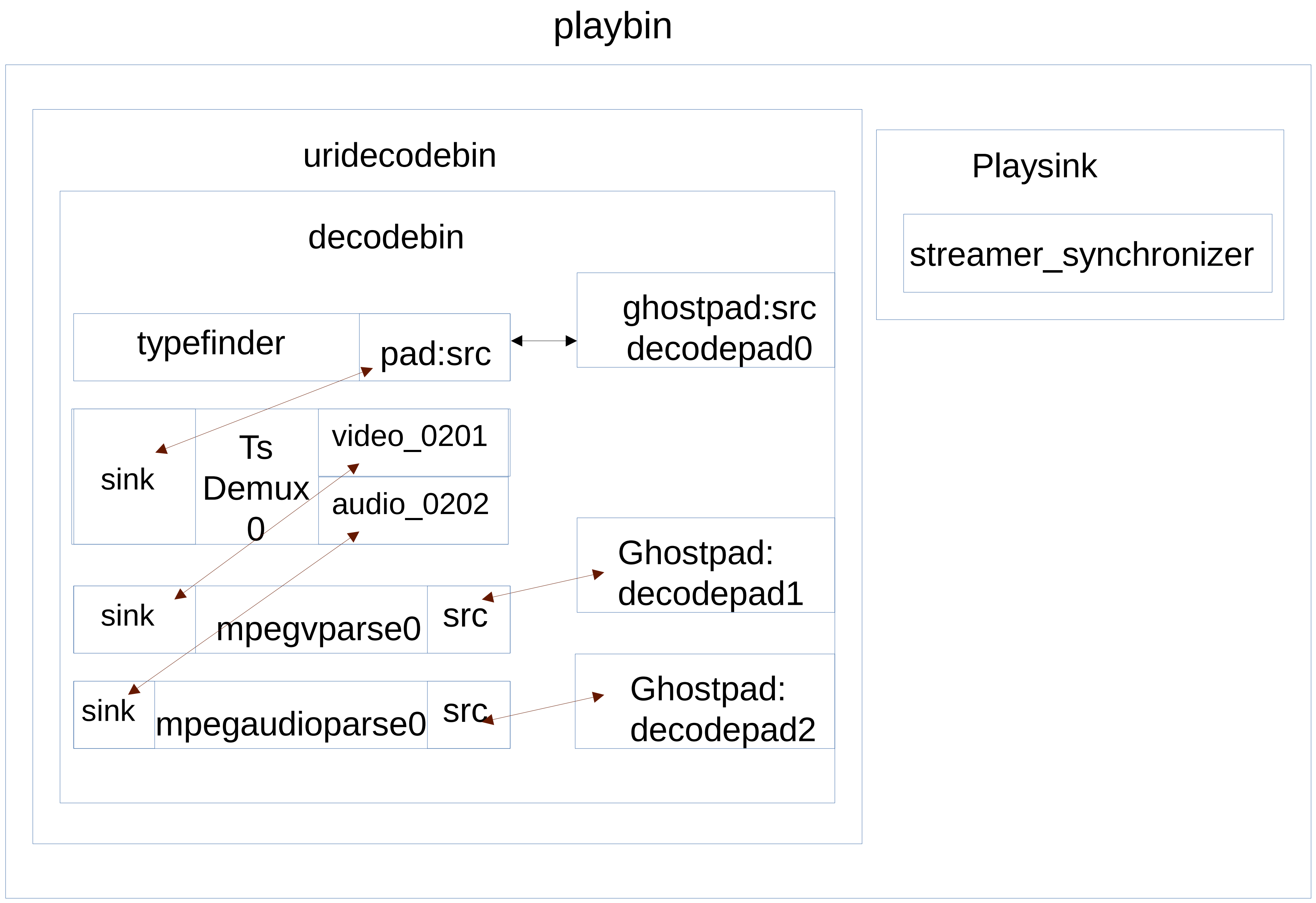
autoplug_query_cb

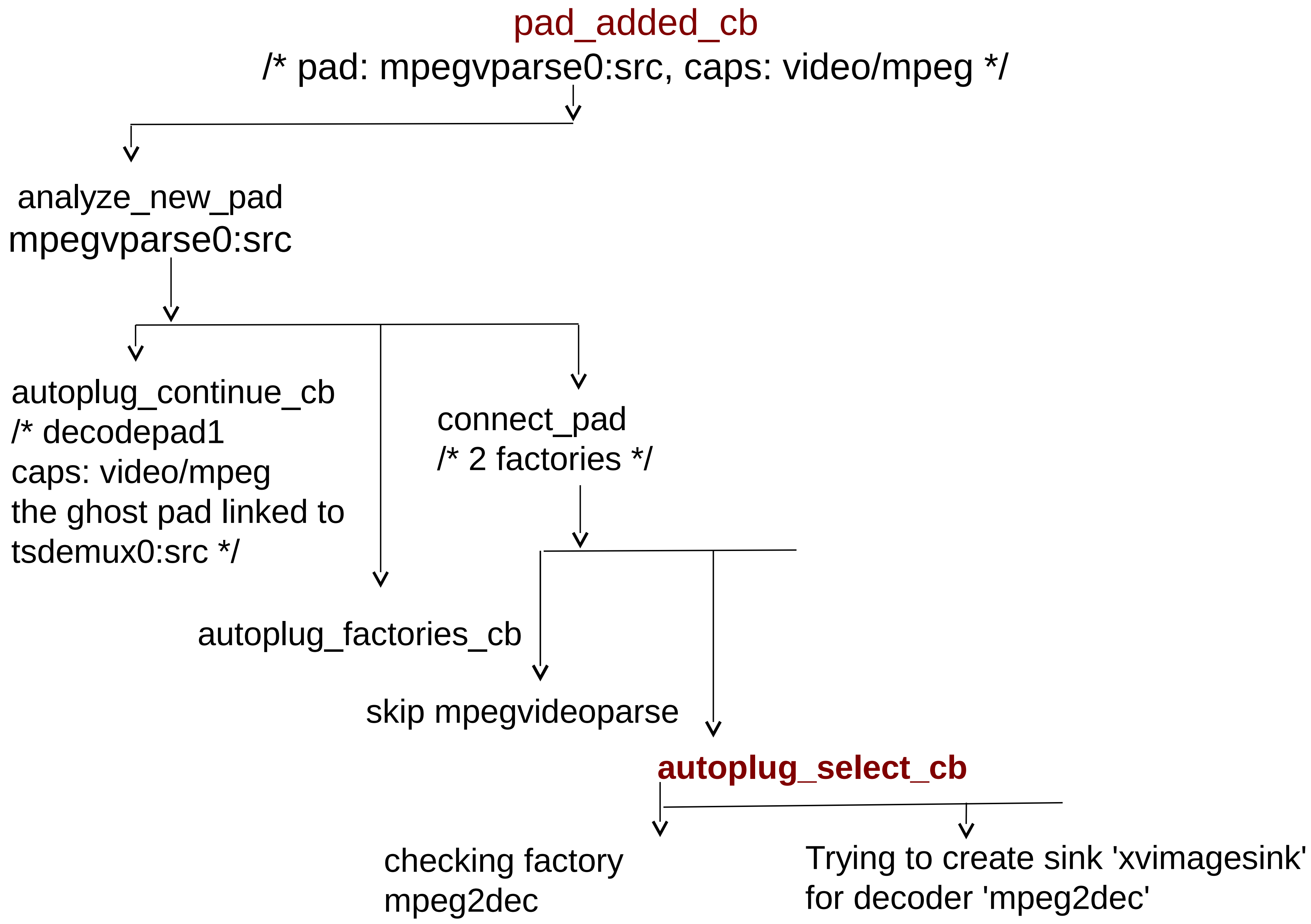
autoplug_query_caps

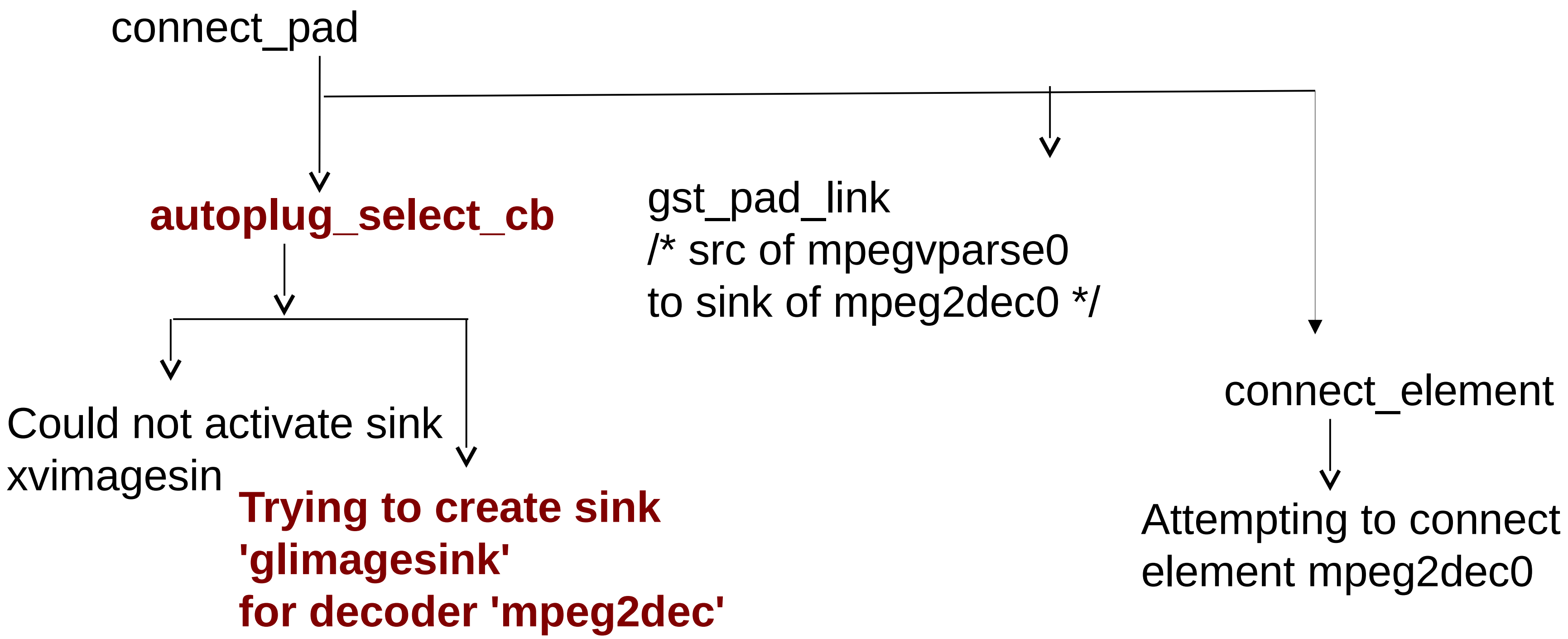
factories =
autoplug_factories_cb
(uridecodebin, pad,NULL,);


playbin
uridecodebin
decodebin
typefinder
pad:src
ghostpad:src
decodepad0
sink
tsdemux0
video_0201
Ghostpad
decodepad1
sink
mpegvparse0
src
Playsink
streamer_synchronizer


pad_added_cb
/* element: decodebin0; pad: tsdemux0:audio_0202; cap: audio/mpeg */
analyze_new_pad
decodebin0
pad tsdemux0:audio_0202
caps:audio/mpeg
gst_decode_pad_new
/* decodepad2
ghost,
cap: audio/mpeg */
autoplug_factories_cb
autoplug_continue_cb
connect_pad
autoplug_select_cb
/* decodepad2
checking factory
mpegaudioparse*/
gst_pad_link
/* link srcpad of tsdemux0
to sinkpad of mpegaudiopase0
*/
connect_element
/* link to mpegaudioparse0
further */


playbin
uridecodebin
decodebin
typefinder
pad:src
ghostpad:src
decodepad0
sink
Ts
Demux
0
video_0201
audio_0202
sink
mpegvparse0
src
Ghostpad:
decodepad1
sink
mpegaudioparse0
src
Ghostpad:
decodepad2
Playsink
streamer_synchronizer


pad_added_cb
/* pad: mpegvparse0:src, caps: video/mpeg */
analyze_new_pad
mpegvparse0:src
autoplug_continue_cb
/* decodepad1
caps: video/mpeg
the ghost pad linked to
tsdemux0:src */
connect_pad
/* 2 factories */
autoplug_factories_cb
skip mpegvideoparse
autoplug_select_cb
checking factory
mpeg2dec
Trying to create sink 'xvimagesink'
for decoder 'mpeg2dec'



connect_pad

autoplug_select_cb

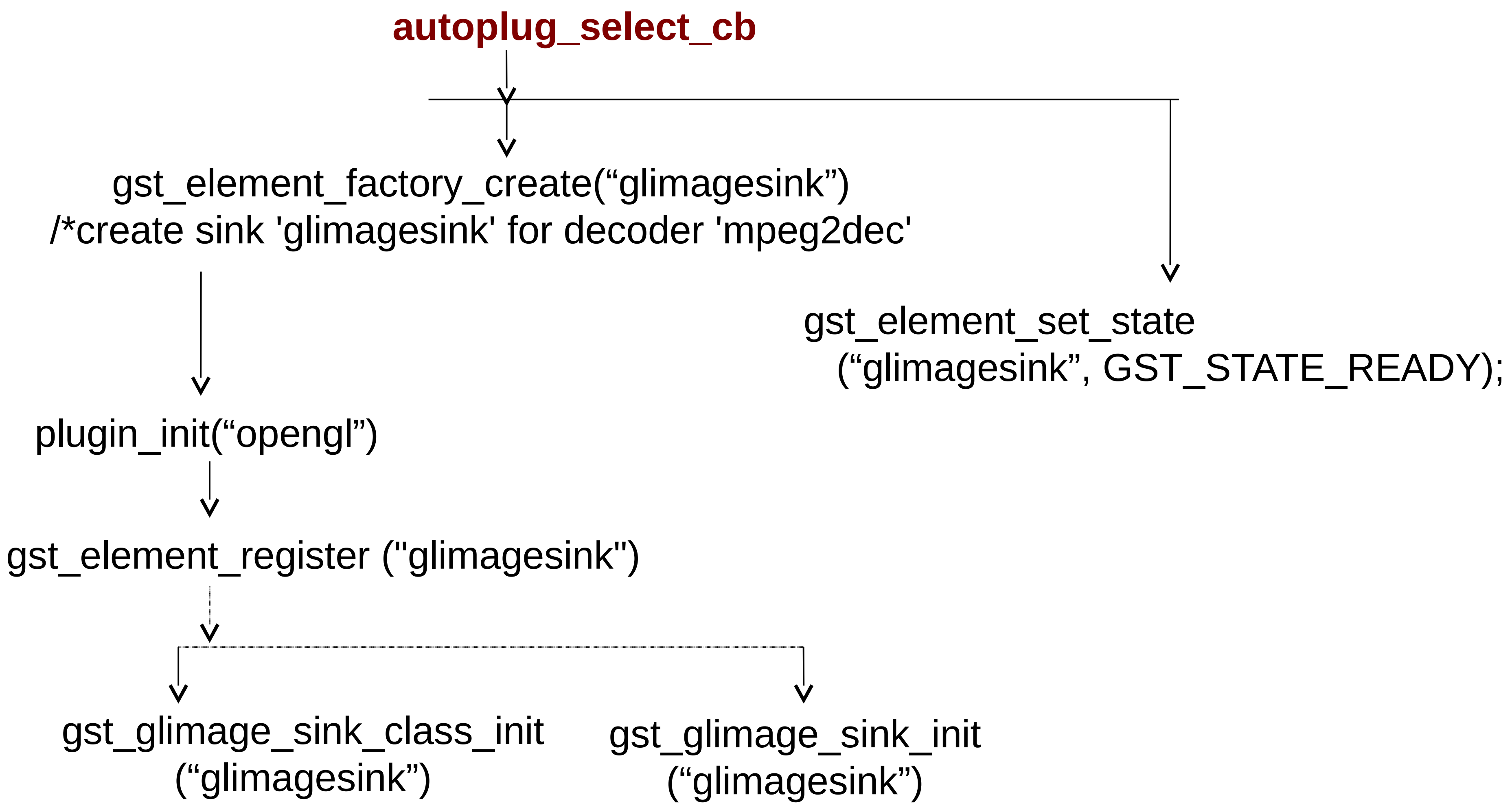
Could not activate sink xvimagesin

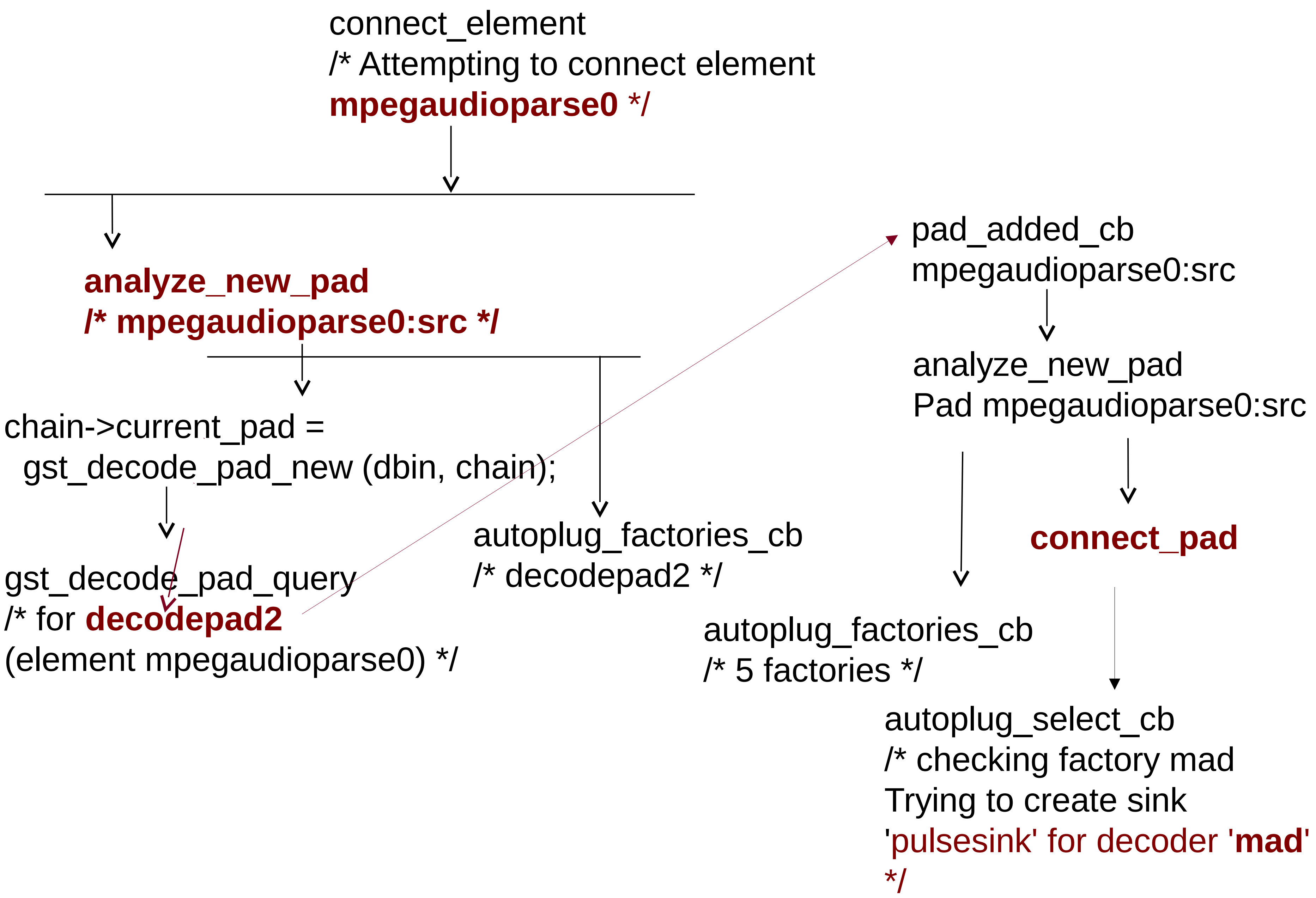
Trying to create sink 'glimagesink' for decoder 'mpeg2dec'

gst_pad_link
/* src of mpegvparse0 to sink of mpeg2dec0 */

connect_element

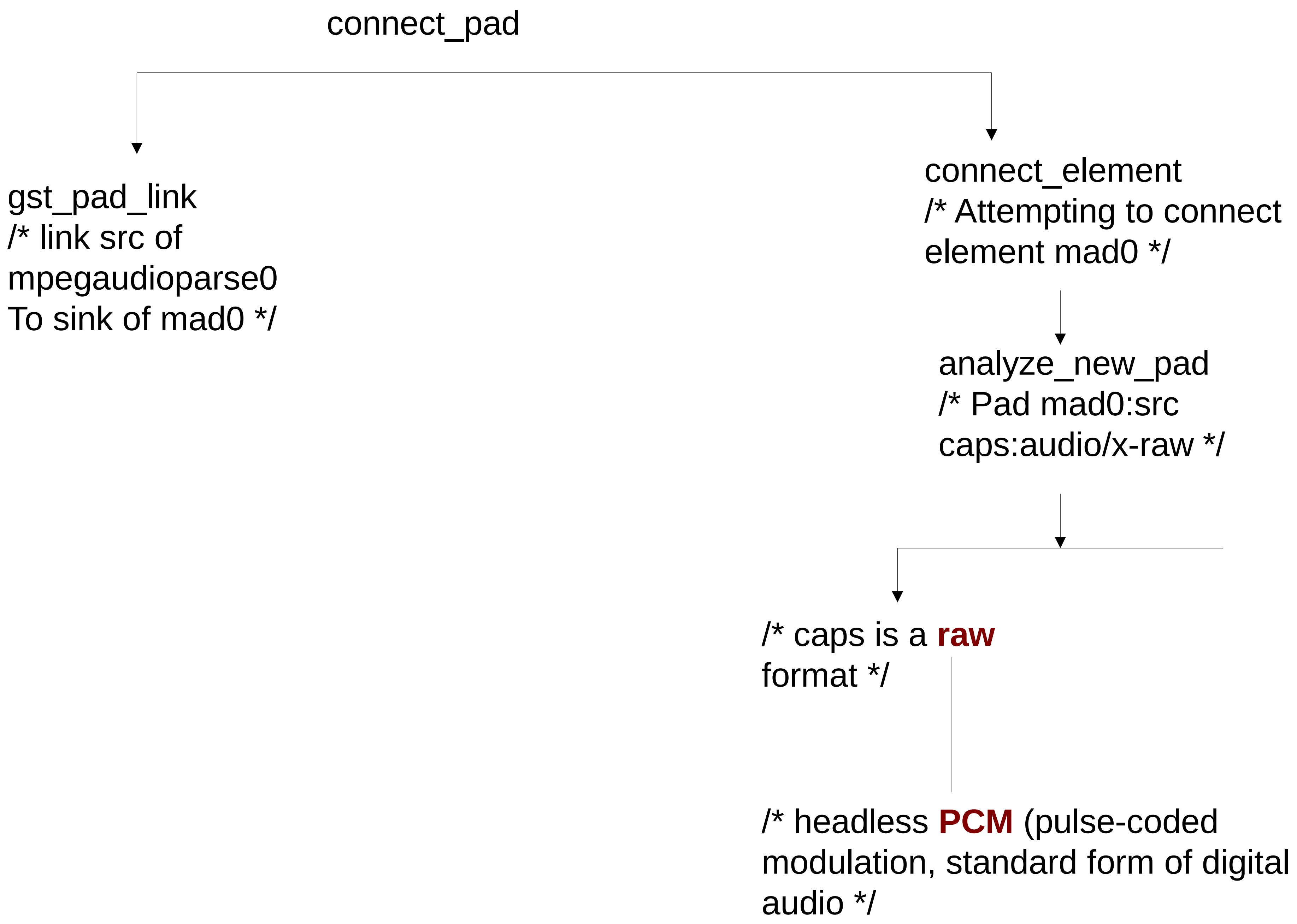
Attempting to connect element mpeg2dec0


autoplug_select_cb
gst_element_factory_create(“glimagesink”)
/*create sink 'glimagesink' for decoder 'mpeg2dec'
gst_element_set_state
(“glimagesink”, GST_STATE_READY);
plugin_init(“opengl”)
gst_element_register ("glimagesink")
gst_glimage_sink_class_init
(“glimagesink”)
gst_glimage_sink_init
(“glimagesink”)


connect_element
/* Attempting to connect element
mpegaudioparse0 */
analyze_new_pad
/* mpegaudioparse0:src */
chain->current_pad =
gst_decode_pad_new (dbin, chain);
gst_decode_pad_query
/* for decodepad2
(element mpegaudioparse0) */
autoplug_factories_cb
/* decodepad2 */
pad_added_cb
mpegaudioparse0:src
analyze_new_pad
Pad mpegaudioparse0:src
connect_pad
autoplug_factories_cb
/* 5 factories */
autoplug_select_cb
/* checking factory mad
Trying to create sink
'pulsesink' for decoder 'mad'
*/

autoplug_select_c
b

```
/* If the sink supports raw audio/video, we first check
 * if the decoder could output any raw audio/video format
 * and assume it is compatible with the sink then. We don't
 * do a complete compatibility check here if converters
 * are plugged between the decoder and the sink because
 * the converters will convert between raw formats and
 * even if the decoder format is not supported by the
decoder
 * a converter will convert it.
 *
 * We assume here that the converters can convert between
 * any raw format.
 */
/* So the audio_sink and video_sink are set into group->video_sink
and group->audio_sink. */
```



connect_pad

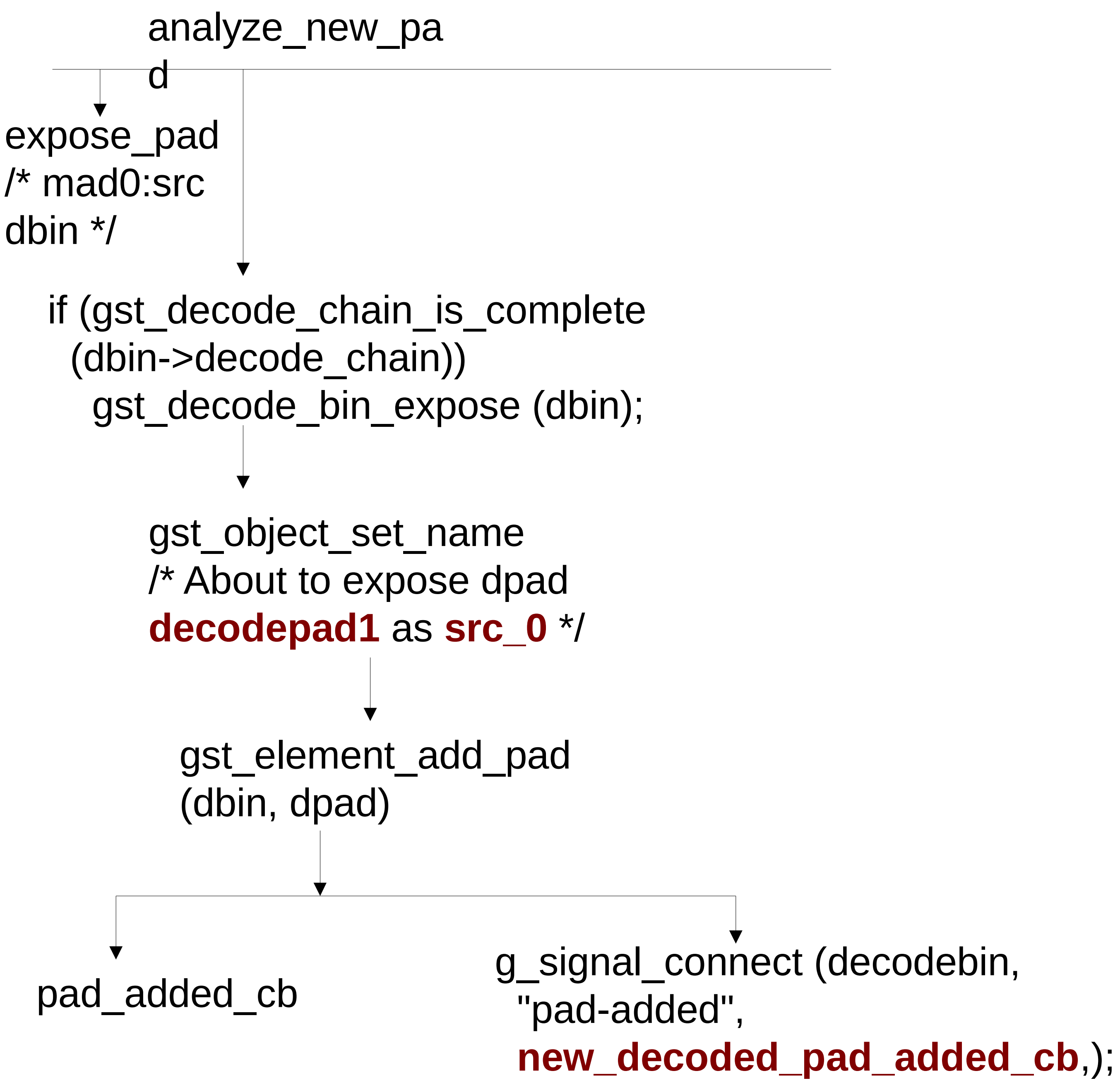
gst_pad_link
/* link src of
mpegaudioparse0
To sink of mad0 */

connect_element
/* Attempting to connect
element mad0 */

analyze_new_pad
/* Pad mad0:src
caps:audio/x-raw */

/* caps is a **raw**
format */

/* headless **PCM** (pulse-coded
modulation, standard form of digital
audio */


analyze_new_pad

expose_pad
/* mad0:src dbin */

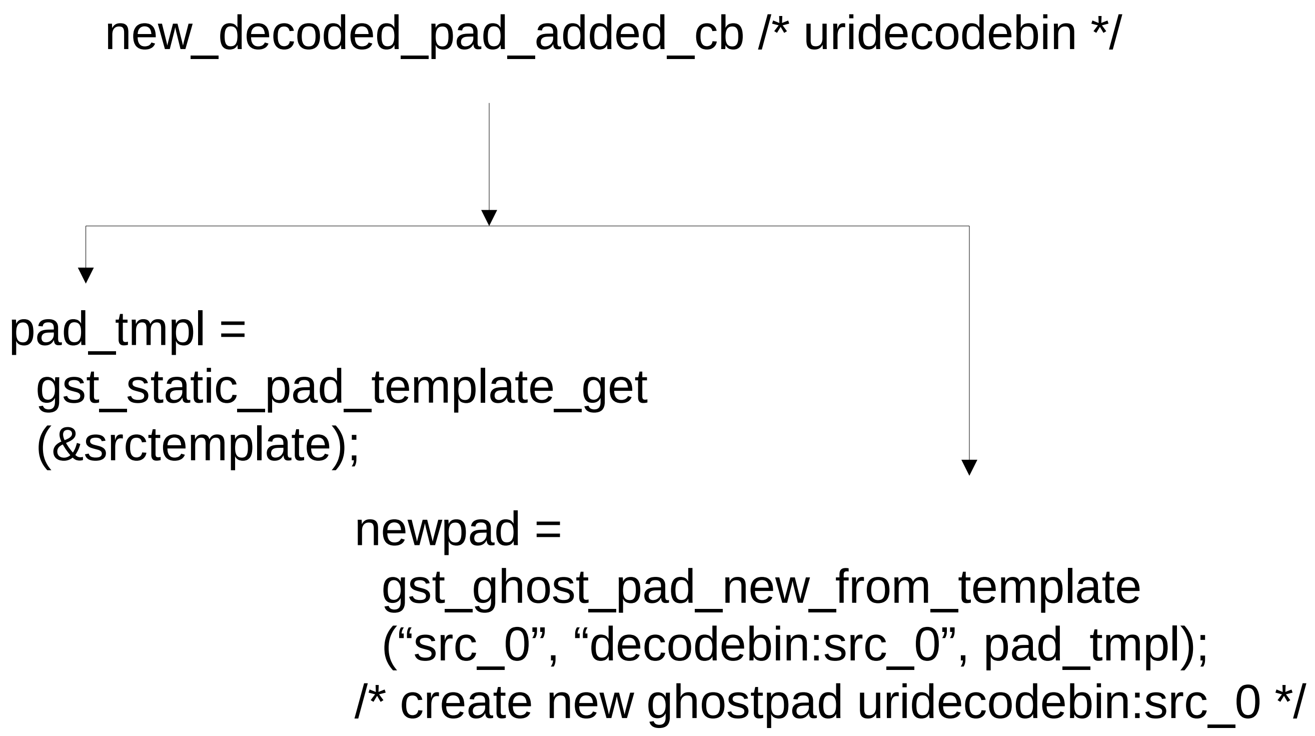
if (gst_decode_chain_is_complete
  (dbin->decode_chain))
    gst_decode_bin_expose (dbin);

gst_object_set_name
/* About to expose dpad
decodepad1 as **src_0** */

gst_element_add_pad
(dbin, dpad)

pad_added_cb

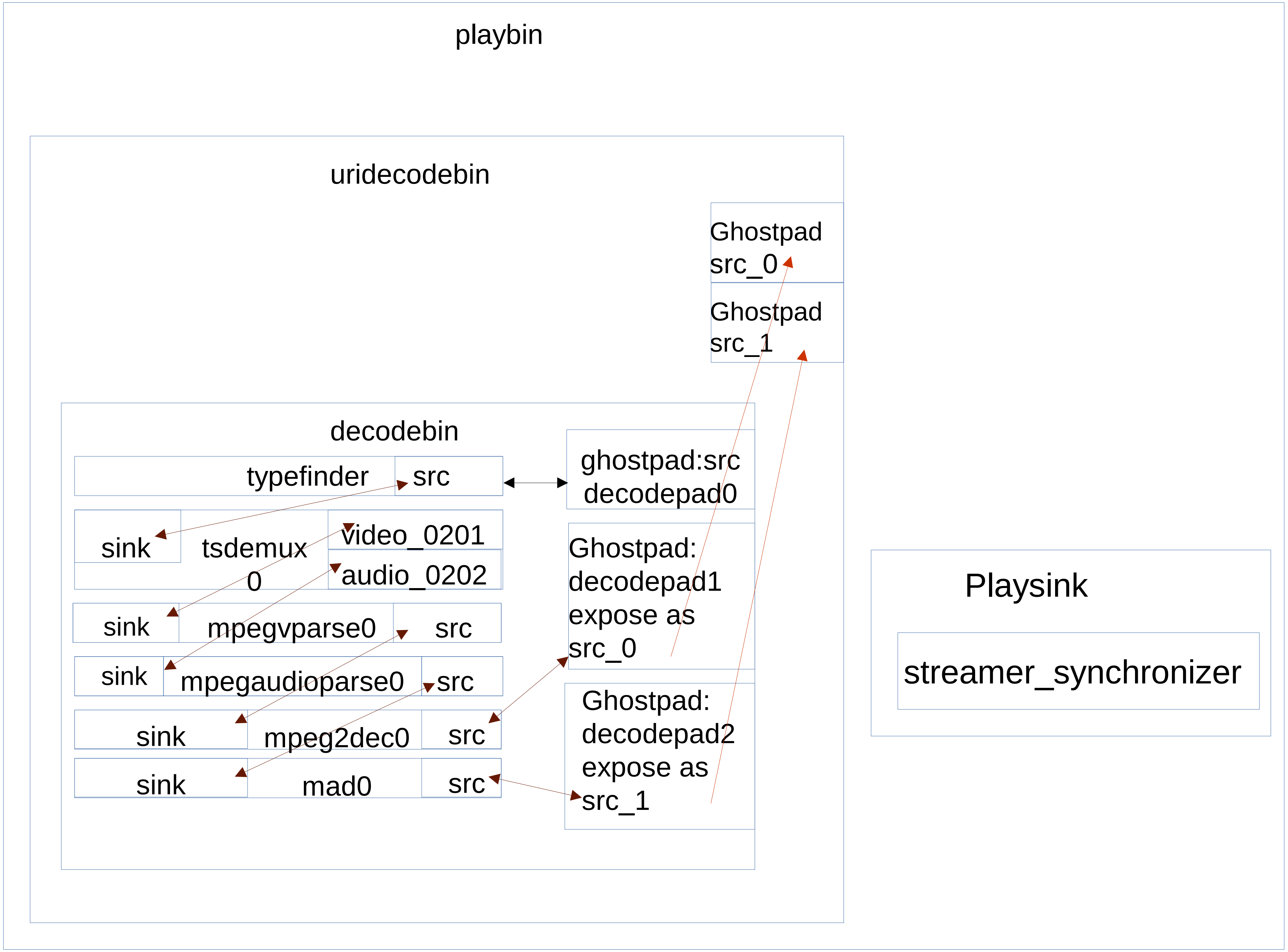
g_signal_connect (decodebin,
  "pad-added",
  new_decoded_pad_added_cb,);



```
new_decoded_pad_added_cb /* uridecodebin */
```

```
pad_tmpl =
  gst_static_pad_template_get
  (&srctemplate);
```

```
newpad =
  gst_ghost_pad_new_from_template
  (“src_0”, “decodebin:src_0”, pad_tmpl);
/* create new ghostpad uridecodebin:src_0 */
```



playbin
uridecodebin
Ghostpad src_0
Ghostpad src_1
decodebin
typefinder
src
ghostpad:src decodepad0
sink
tsdemux 0
video_0201
audio_0202
Ghostpad: decodepad1 expose as src_0
sink
mpegvparse0
src
sink
mpegaudioparse0
src
sink
mpeg2dec0
src
Ghostpad: decodepad2 expose as src_1
sink
mad0
src
Playsink
streamer_synchronizer

pad_added_cb /* playbin */

combine->combiner
  = gst_element_factory_make
  ("input-selector", NULL);
/* creating new input selector */

gst_bin_add
  (playbin,
  combine->combiner);
/* adding new stream combiner,
(srcpad). It's possible that
multiple decodebins push data
into the combiner */

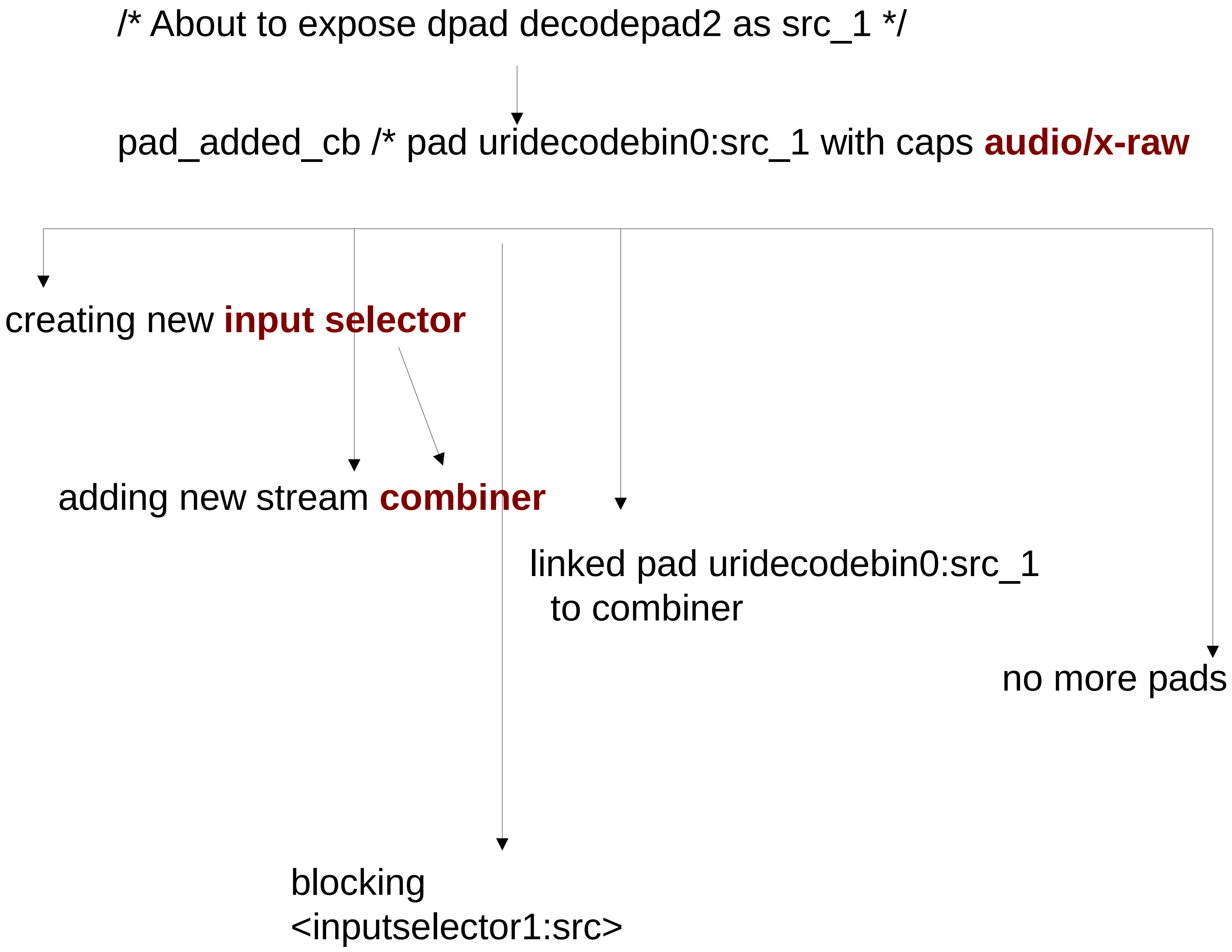
blocking
<**inputselector0:src**>

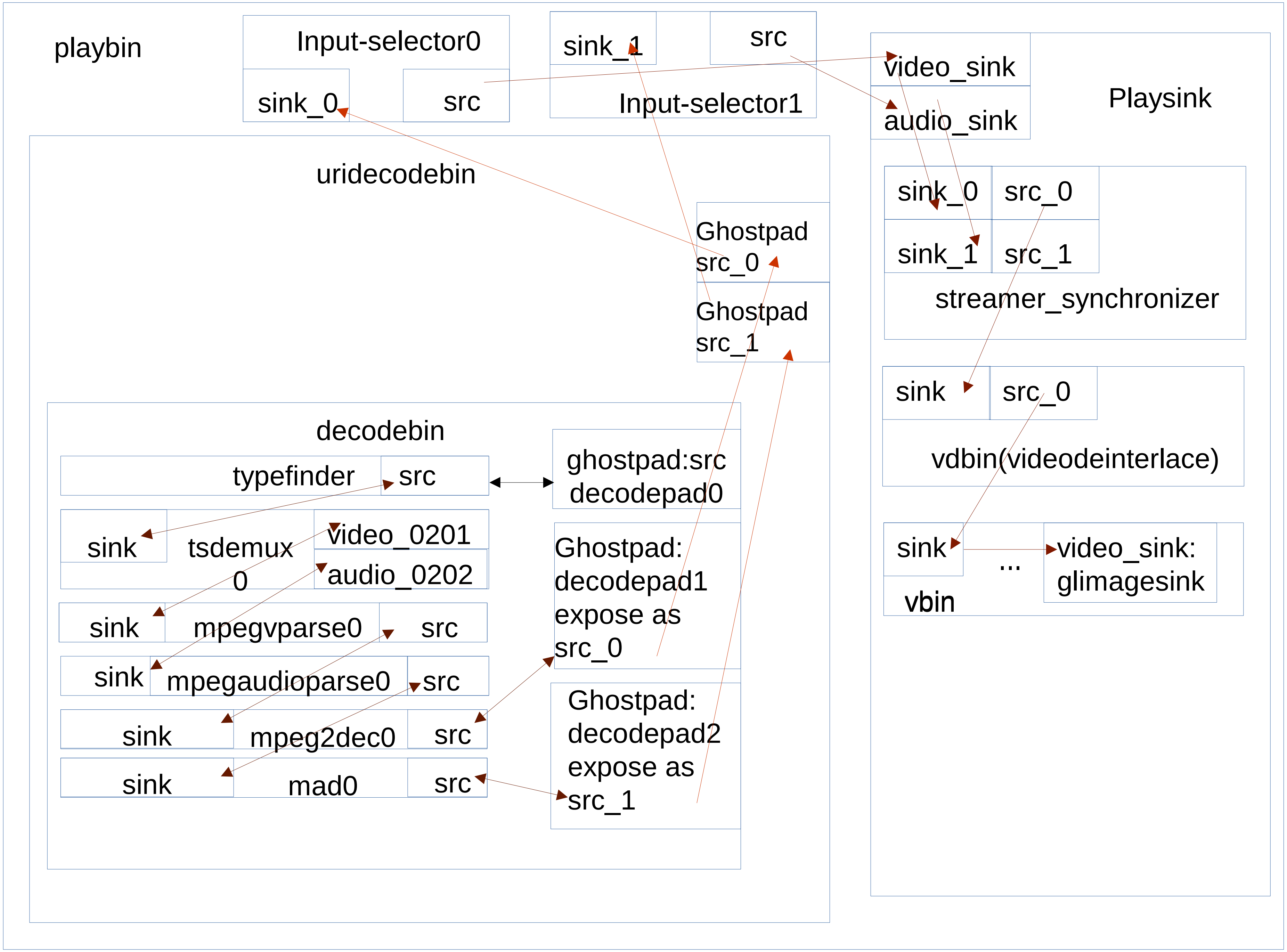
sinkpad =
  gst_element_get_request_pad
  (combine->combiner, "sink_%u")
/* get sinkpad for the new stream */

got pad
inputselector0:sink_0
from combiner

linked pad uridecodebin0:src_0
  to combiner

gst_element_no_more_pads
/* signaling **no-more-pads** */


/* About to expose dpad decodepad2 as src_1 */
pad_added_cb /* pad uridecodebin0:src_1 with caps **audio/x-raw**
creating new **input selector**
adding new stream **combiner**
linked pad uridecodebin0:src_1 to combiner
no more pads
blocking <inputselector1:src>


playbin
Input-selector0
sink_0
src
sink_1
src
Input-selector1
video_sink
audio_sink
Playsink
uridecodebin
Ghostpad src_0
Ghostpad src_1
sink_0
src_0
sink_1
src_1
streamer_synchronizer
sink
src_0
vdbin(videodeinterlace)
decodebin
typefinder
src
ghostpad:src decodepad0
sink
tsdemux 0
video_0201
audio_0202
sink
mpegvparse0
src
sink
mpegaudioparse0
src
sink
mpeg2dec0
src
sink
mad0
src
Ghostpad: decodepad1 expose as src_0
Ghostpad: decodepad2 expose as src_1
sink
...
video_sink: glimagesink
vbin

no_more_pads_cb
/* called when all pads are available and we must connect the sinks to them.*/

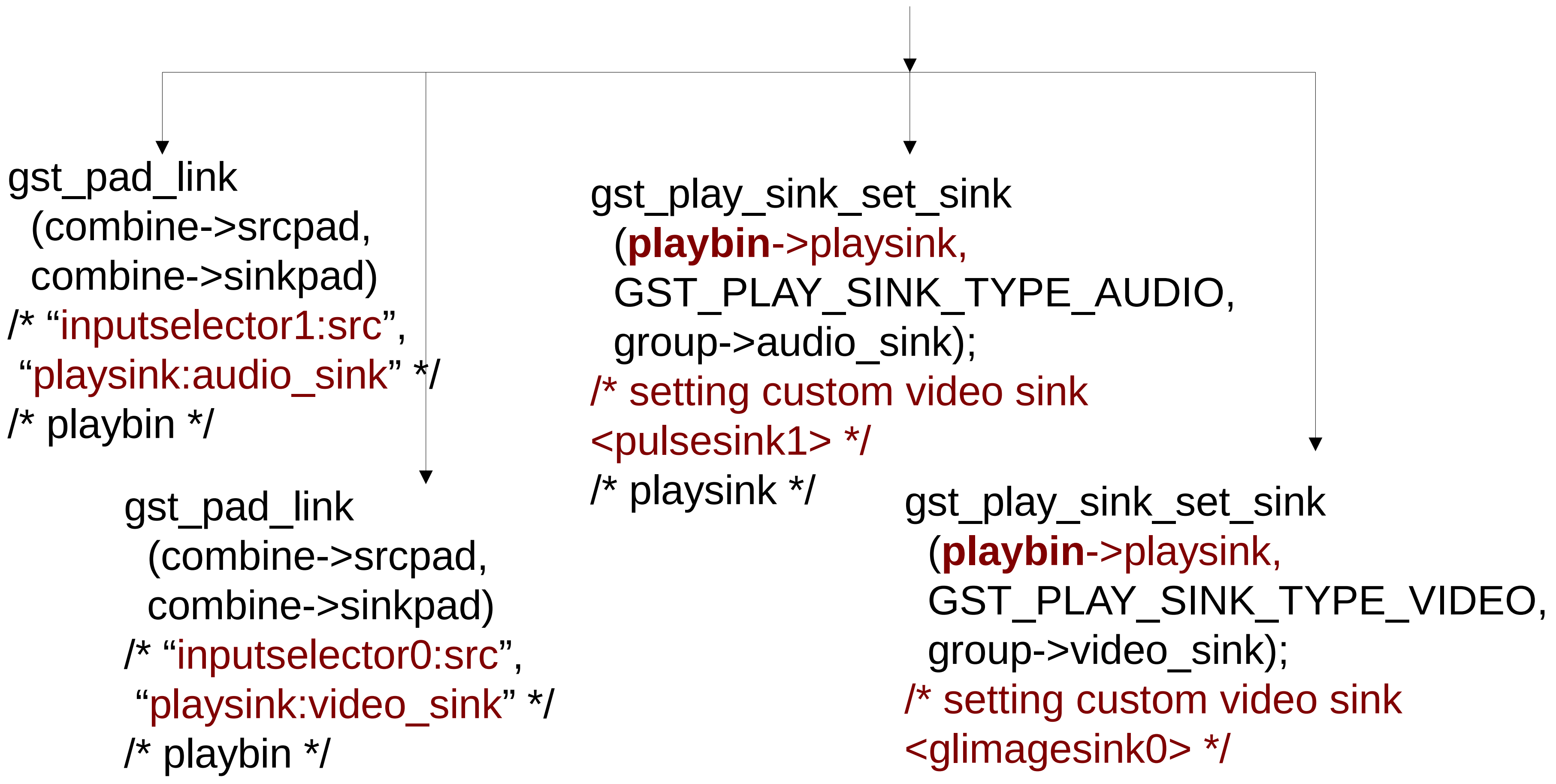
requesting new sink
request pad type 0

```
playsink->audio_tee =
  gst_element_factory_make
  ("tee", "audiotee");
```

```
playsink->audio_pad =
  gst_ghost_pad_new
  (, playsink->audio_tee_sink);
```

“playsink:audio_sink”   “audiotee:sink”

no_more_pads_cb
/* called when all pads are available and we must connect the sinks to them.*/

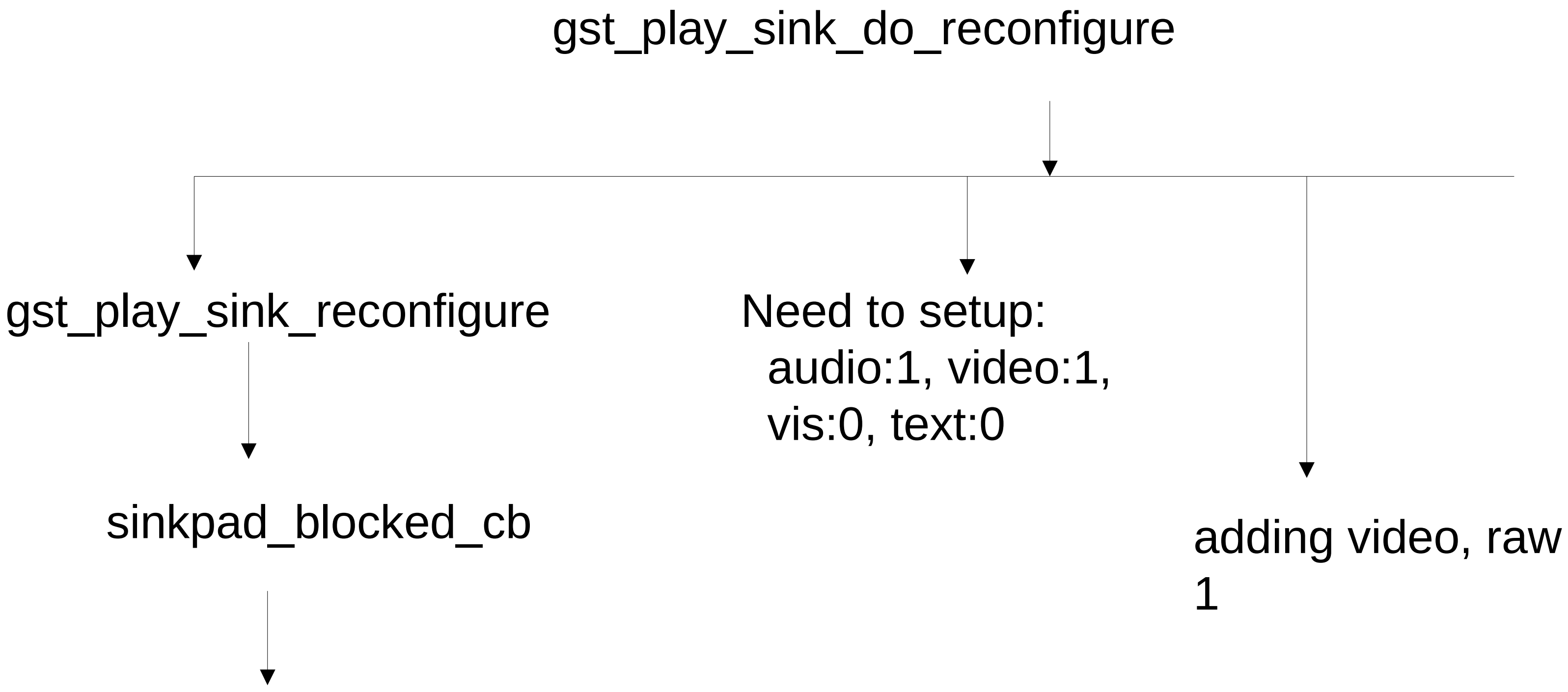


```
gst_pad_link
  (combine->srcpad,
  combine->sinkpad)
/* “inputselector1:src”,
 “playsink:audio_sink” */
/* playbin */
```

```
gst_pad_link
  (combine->srcpad,
  combine->sinkpad)
/* “inputselector0:src”,
 “playsink:video_sink” */
/* playbin */
```

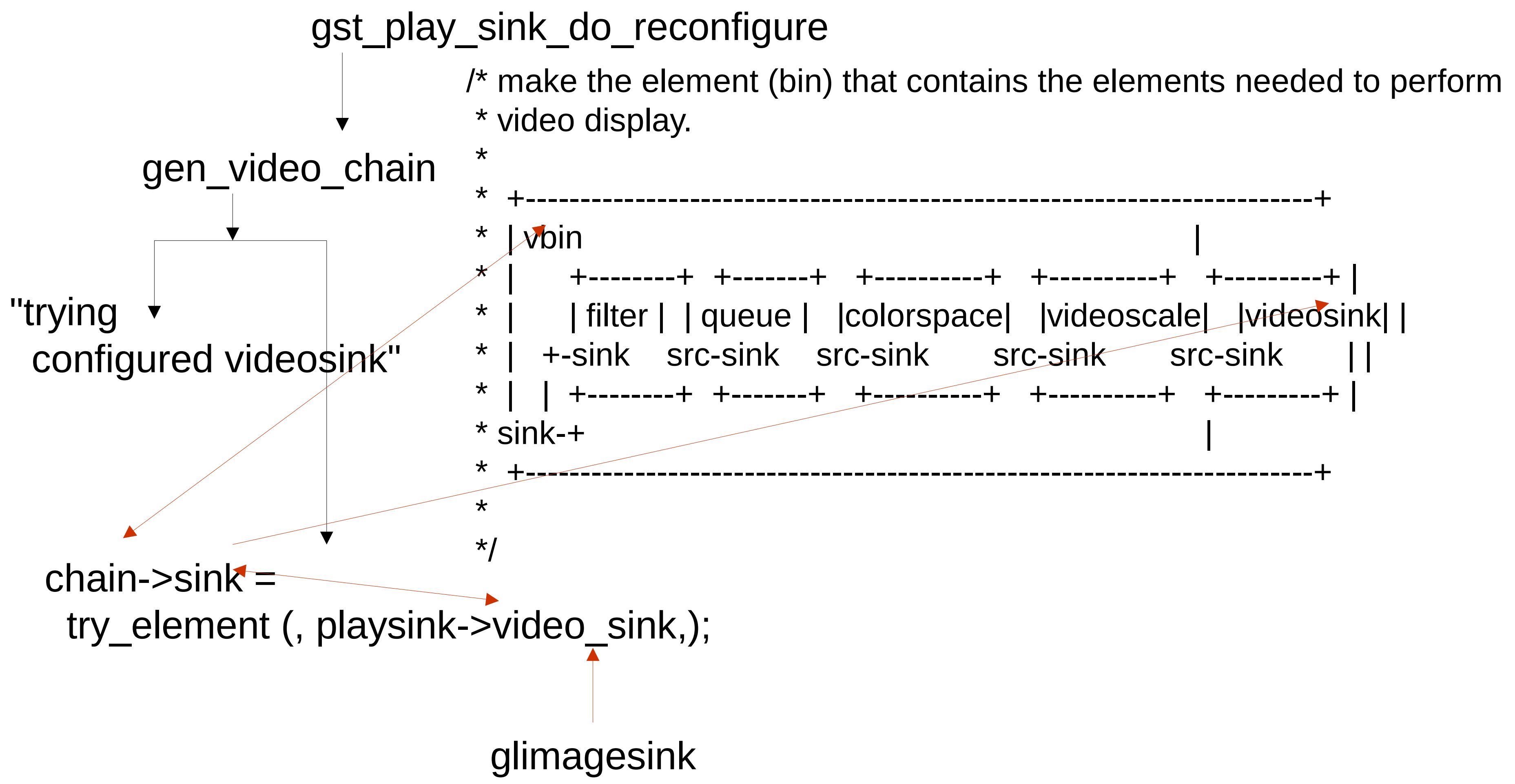
```
gst_play_sink_set_sink
  (playbin->playsink,
  GST_PLAY_SINK_TYPE_AUDIO,
  group->audio_sink);
/* setting custom video sink
<pulsesink1> */
/* playsink */
```

```
gst_play_sink_set_sink
  (playbin->playsink,
  GST_PLAY_SINK_TYPE_VIDEO,
  group->video_sink);
/* setting custom video sink
<glimagesink0> */
```


gst_play_sink_do_reconfigure
gst_play_sink_reconfigure
Need to setup: audio:1, video:1, vis:0, text:0
adding video, raw 1
sinkpad_blocked_cb

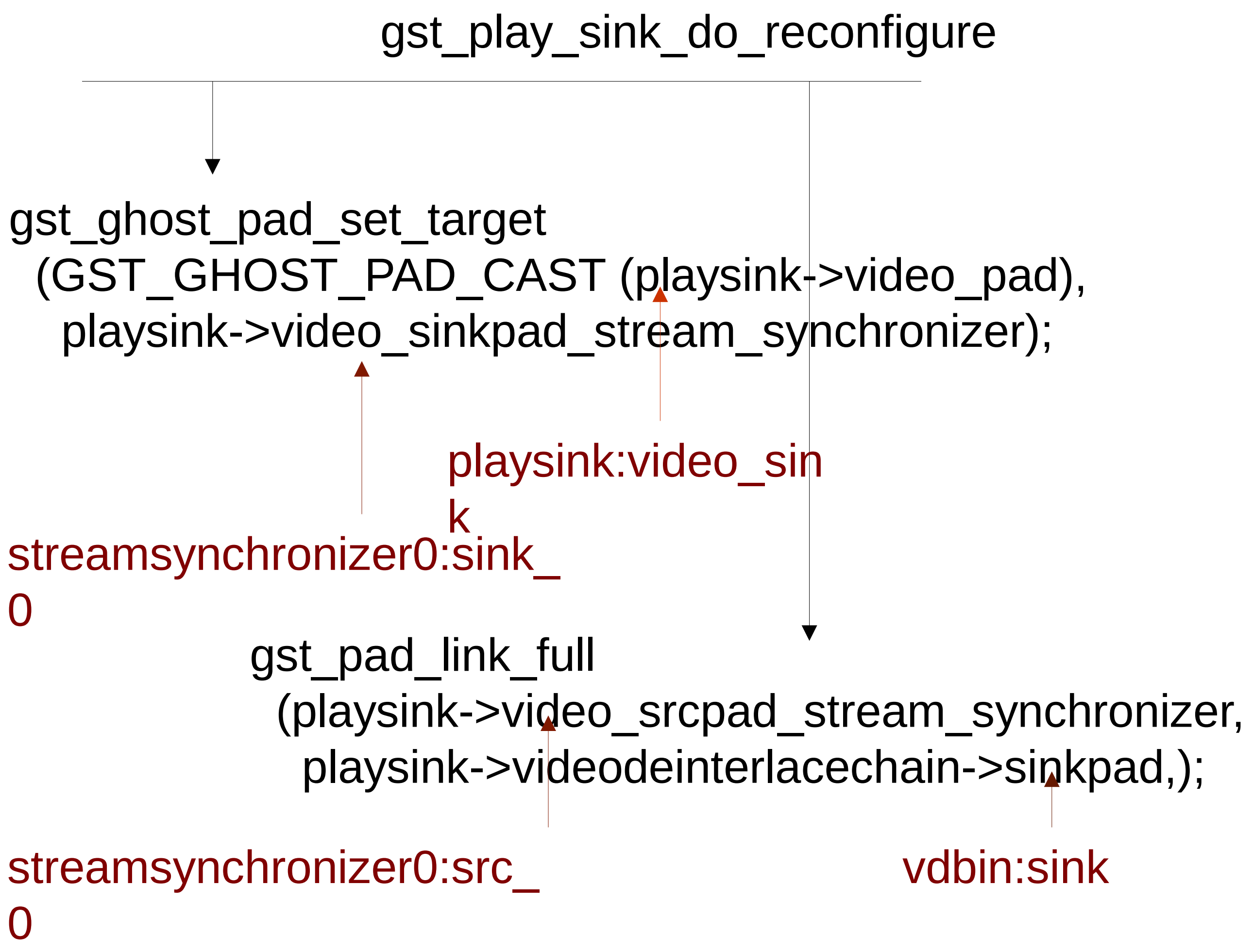
gst_play_sink_do_reconfigure
/* called when all the request pads
are requested and when we have
to construct the final pipeline.*/

/* playsink.video_sink
= group->video_sink
= glimagesink */



```
/* make the element (bin) that contains the elements needed to perform
 * video display.
 *
 *  +------------------------------------------------------------------------+
 *  | vbin                                                                   |
 *  |      +--------+   +-------+   +----------+   +----------+   +---------+ |
 *  |      | filter |   | queue |   |colorspace|   |videoscale|   |videosink| |
 *  |   +-sink    src-sink    src-sink        src-sink        src-sink       | |
 *  |   |  +--------+   +-------+   +----------+   +----------+   +---------+ |
 * sink-+                                                                    |
 *  +------------------------------------------------------------------------+
 *
 */
```

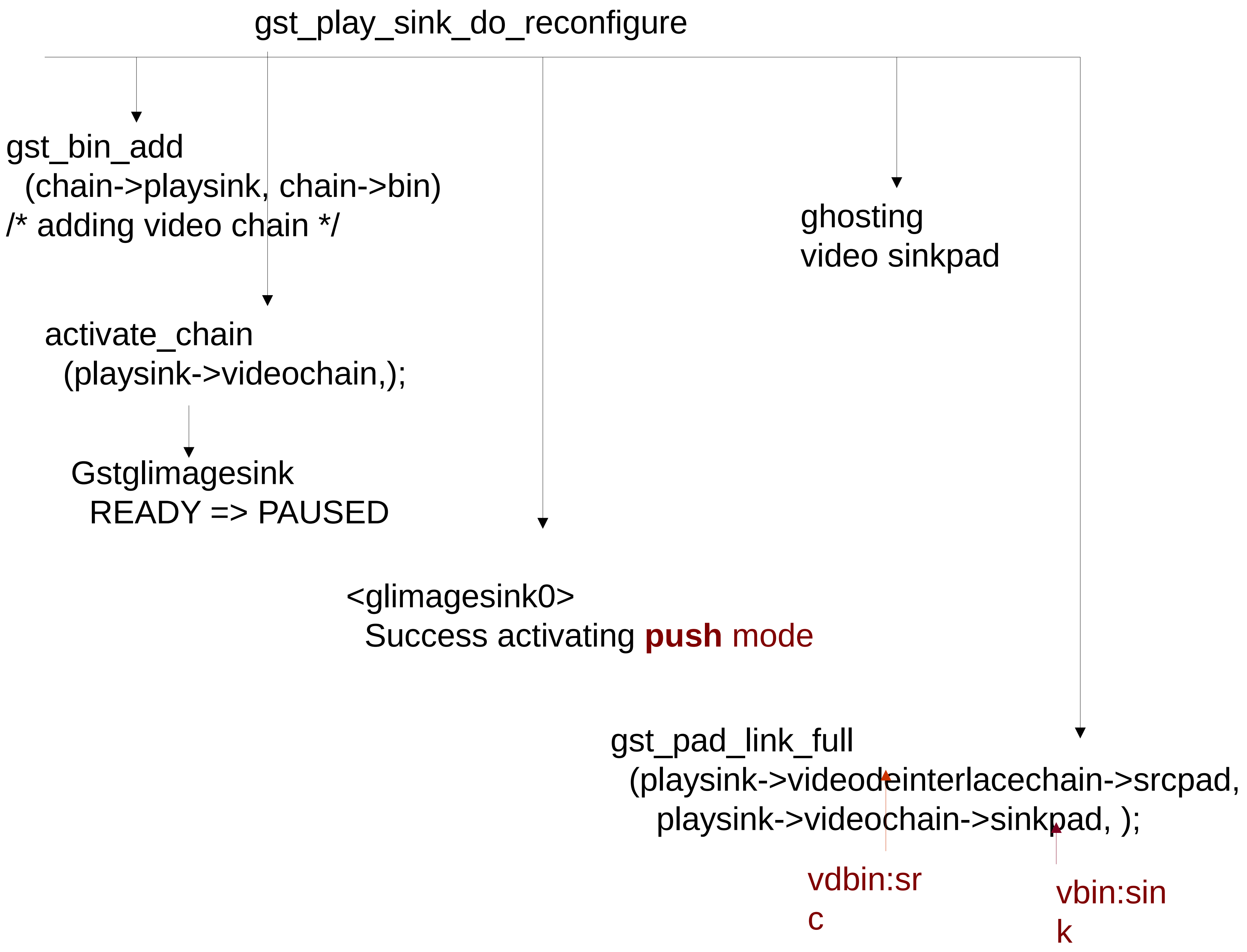
```
chain->sink =
  try_element (, playsink->video_sink,);
```



gst_play_sink_do_reconfigure

```
gst_ghost_pad_set_target
  (GST_GHOST_PAD_CAST (playsink->video_pad),
    playsink->video_sinkpad_stream_synchronizer);
```

```
gst_pad_link_full
  (playsink->video_srcpad_stream_synchronizer,
    playsink->videodeinterlacechain->sinkpad,);
```



gst_play_sink_do_reconfigure

gst_bin_add
  (chain->playsink, chain->bin)
/* adding video chain */

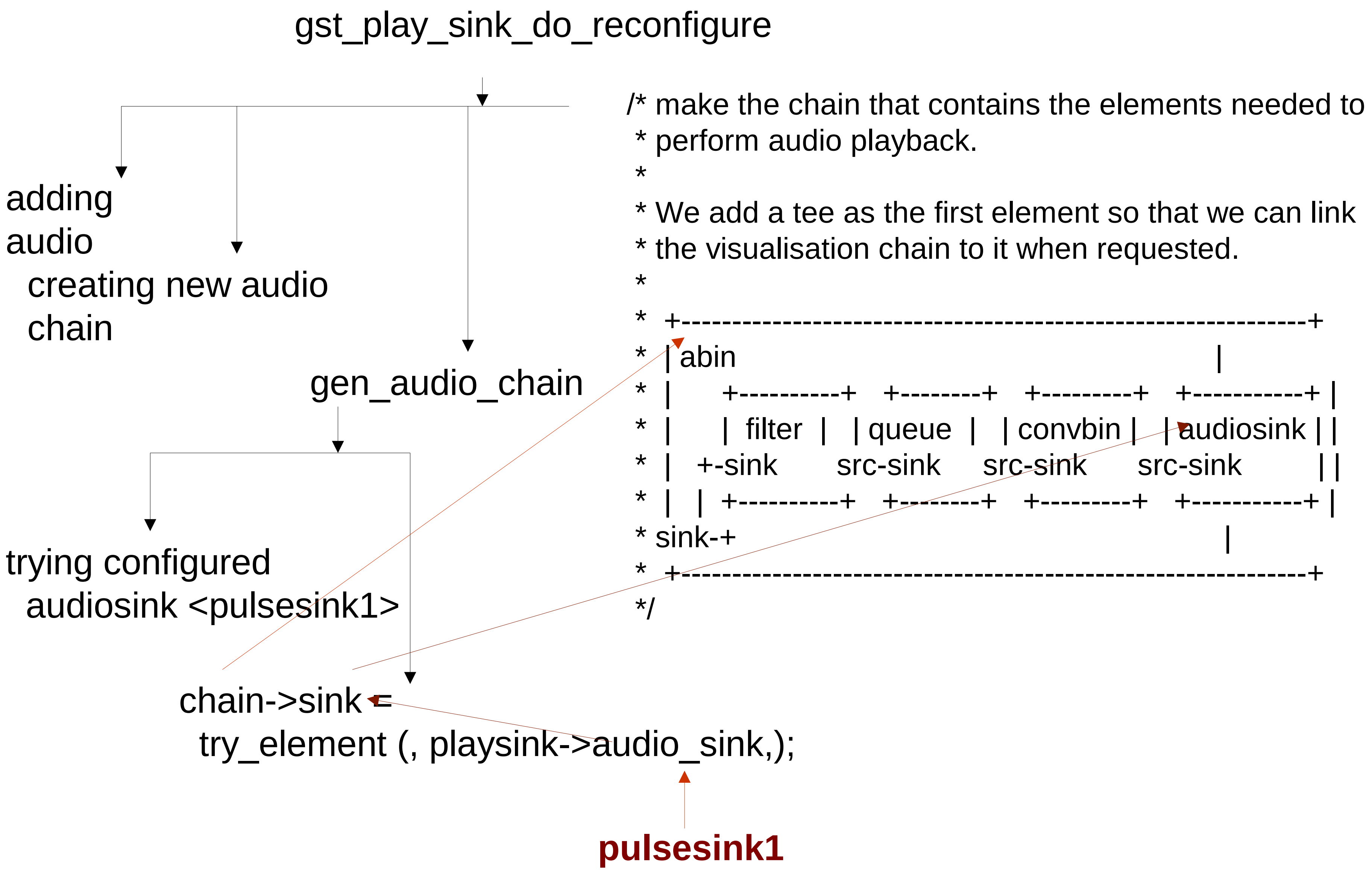
activate_chain
  (playsink->videochain,);

Gstglimagesink
  READY => PAUSED

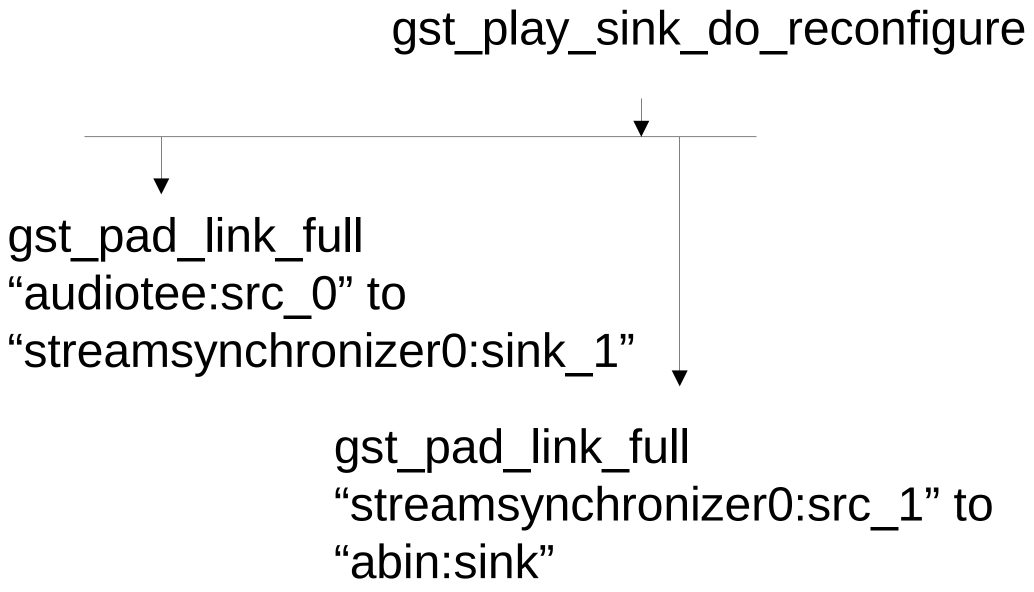
<glimagesink0>
  Success activating **push** mode

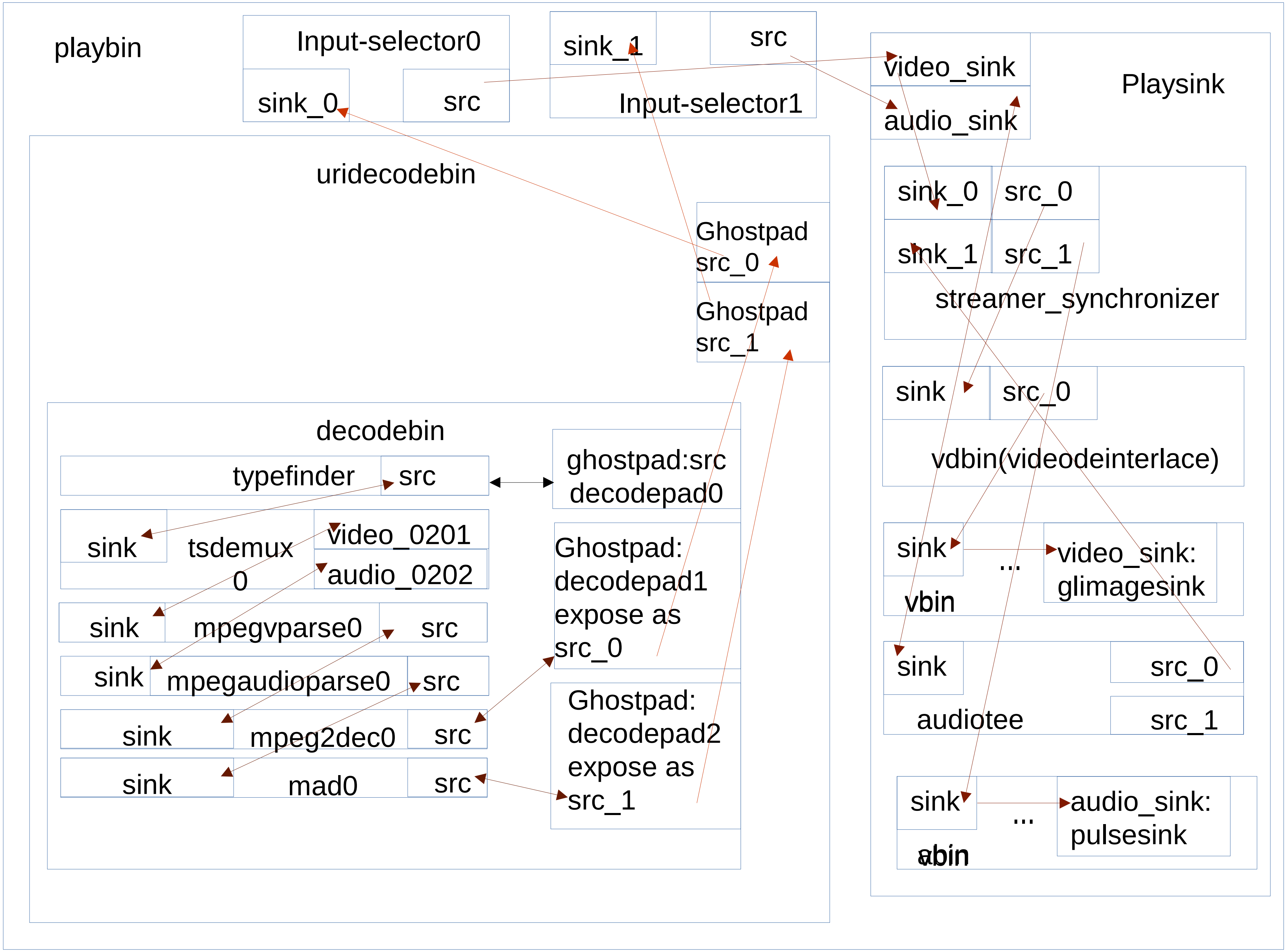
ghosting
video sinkpad

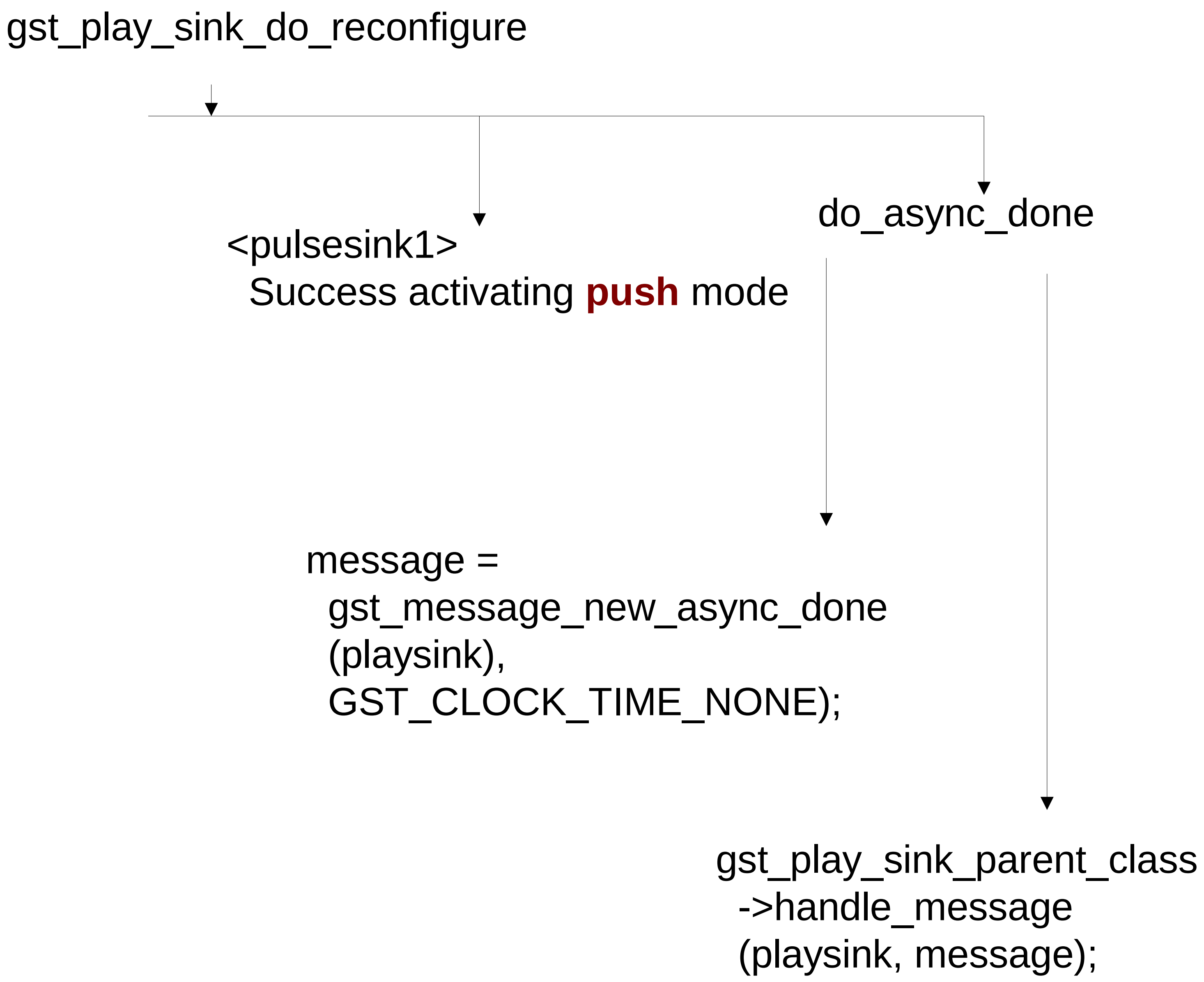
gst_pad_link_full
  (playsink->videodeinterlacechain->srcpad,
    playsink->videochain->sinkpad,);


gst_play_sink_do_reconfigure
adding audio
creating new audio chain
gen_audio_chain
trying configured audiosink <pulsesink1>
chain->sink =
try_element (, playsink->audio_sink,);
pulsesink1

```
/* make the chain that contains the elements needed to
 * perform audio playback.
 *
 * We add a tee as the first element so that we can link
 * the visualisation chain to it when requested.
 *
 *  +-------------------------------------------------------------+
 *  | abin                                                        |
 *  |      +----------+   +--------+   +---------+   +-----------+ |
 *  |      |  filter  |   | queue  |   | convbin |   | audiosink | |
 *  |   +-sink      src-sink     src-sink      src-sink          | |
 *  |   |  +----------+   +--------+   +---------+   +-----------+ |
 * sink-+                                                         |
 *  +-------------------------------------------------------------+
 */
```


gst_play_sink_do_reconfigure
gst_pad_link_full
“audiotee:src_0” to
“streamsynchronizer0:sink_1”
gst_pad_link_full
“streamsynchronizer0:src_1” to
“abin:sink”


playbin
Input-selector0
sink_0
src
sink_1
src
Input-selector1
video_sink
audio_sink
Playsink
uridecodebin
Ghostpad src_0
Ghostpad src_1
sink_0
src_0
sink_1
src_1
streamer_synchronizer
sink
src_0
vdbin(videodeinterlace)
decodebin
typefinder
src
ghostpad:src decodepad0
sink
tsdemux 0
video_0201
audio_0202
sink
mpegvparse0
src
sink
mpegaudioparse0
src
sink
mpeg2dec0
src
sink
mad0
src
Ghostpad: decodepad1 expose as src_0
Ghostpad: decodepad2 expose as src_1
sink
...
video_sink: glimagesink
vbin
sink
src_0
audiotee
src_1
sink
...
audio_sink: pulsesink
abin



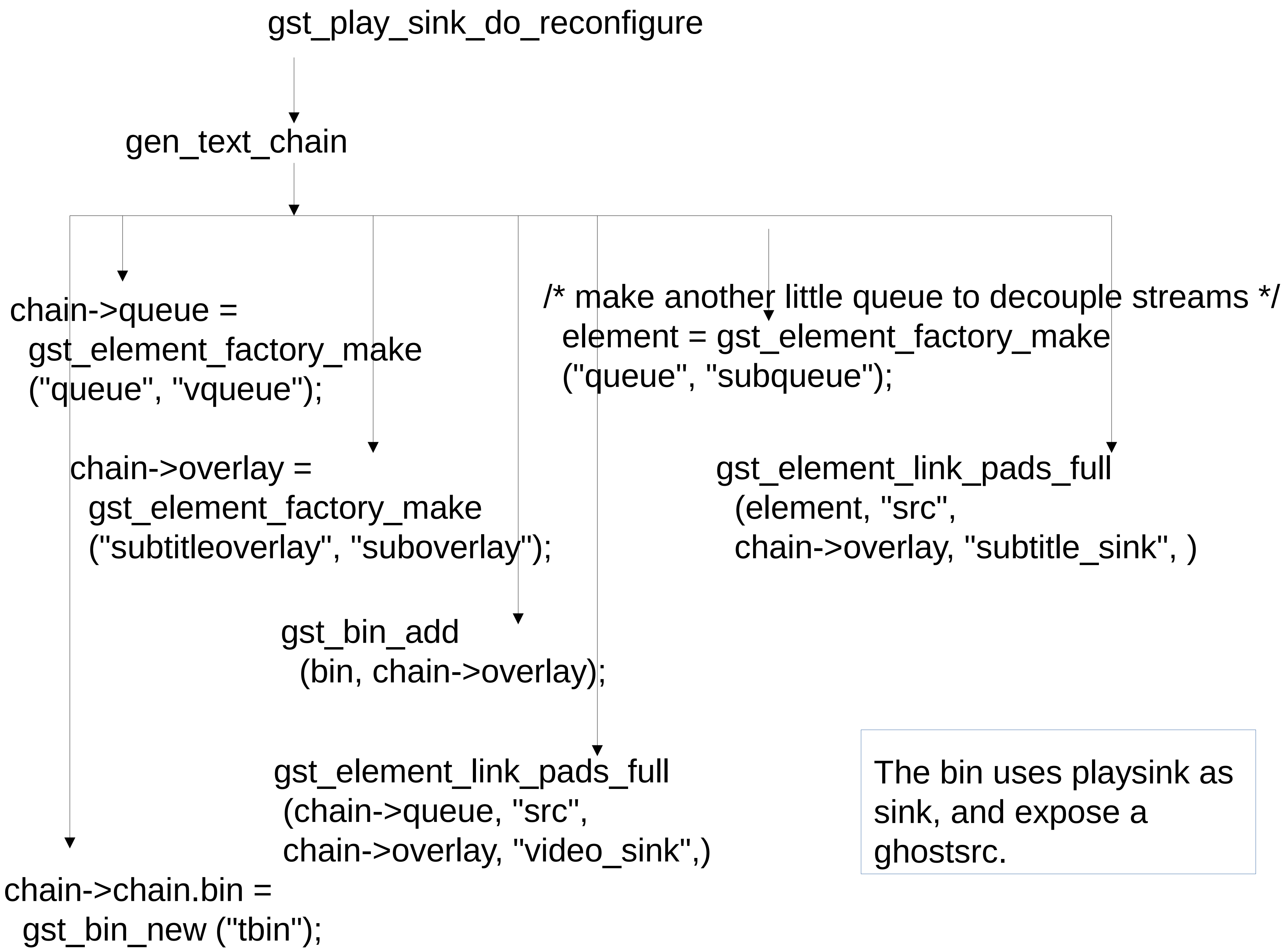
gst_play_sink_do_reconfigure

<pulsesink1>
  Success activating **push** mode

do_async_done

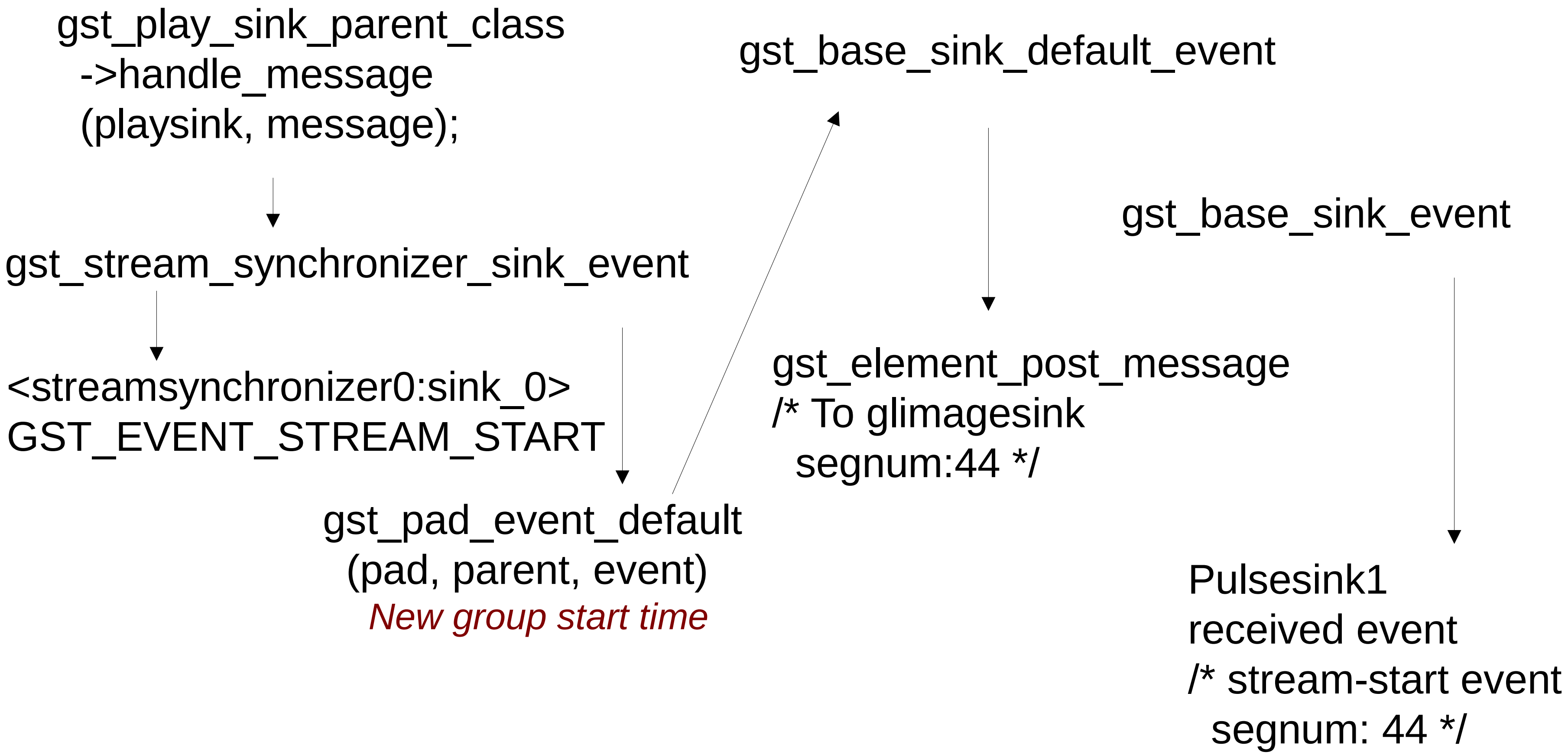
```
message =
  gst_message_new_async_done
  (playsink),
  GST_CLOCK_TIME_NONE);
```

```
gst_play_sink_parent_class
  ->handle_message
  (playsink, message);
```



```
gst_play_sink_do_reconfigure
  gen_text_chain
    chain->chain.bin =
      gst_bin_new ("tbin");
    chain->queue =
      gst_element_factory_make
      ("queue", "vqueue");
    chain->overlay =
      gst_element_factory_make
      ("subtitleoverlay", "suboverlay");
    gst_bin_add
      (bin, chain->overlay);
    gst_element_link_pads_full
     (chain->queue, "src",
     chain->overlay, "video_sink",)
    /* make another little queue to decouple streams */
      element = gst_element_factory_make
      ("queue", "subqueue");
    gst_element_link_pads_full
      (element, "src",
      chain->overlay, "subtitle_sink", )
```

The bin uses playsink as sink, and expose a ghostsrc.


gst_play_sink_parent_class ->handle_message (playsink, message);
gst_stream_synchronizer_sink_event
<streamsynchronizer0:sink_0> GST_EVENT_STREAM_START
gst_pad_event_default (pad, parent, event)
New group start time
gst_base_sink_default_event
gst_element_post_message /* To glimagesink segnum:44 */
gst_base_sink_event
Pulsesink1 received event /* stream-start event segnum: 44 */

Pulsesink1
received event
“stream-start event”
/* gstbasesink.c */

store_sticky_event
<playsink:audio_sink>
”notify caps”
/* gstpad.c */

gst_pulsesink_query_getcaps
/* gstpulsesink.c */

```
   GST_DEBUG_OBJECT (pad, "notify caps");
   g_object_notify_by_pspec ((GObject *) pad,
pspec_caps);
```

caps_notify_cb
/* gstplaysink.c */

gst_stream_synchronizer_sink_event

Stream
start

g_object_notify (, "tags");
/* gstinputselector.c */

notify_tags_cb
<inputselector1:sink_0>
with stream id 0

<pulsesink1>
received caps event
“audio/x-raw”

gst_base_sink_wait_preroll:
<pulsesink1>
waiting in preroll for flush
or PLAYING

configure_stream_bufferin
g
/* uridecodebin */

caps_notify_cb
Video caps changed

gst_glimage_sink_set_caps
set caps with video/x-raw
width=(int)1280, height=(int)720
/* gstglimagesink.c */

notify_tags_cb
pad <inputselector0:sink_0>
with stream id 0 and type 2

commiting state to
PAUSED

posting async-done
message

app says:
Setting pipeline to
PLAYING