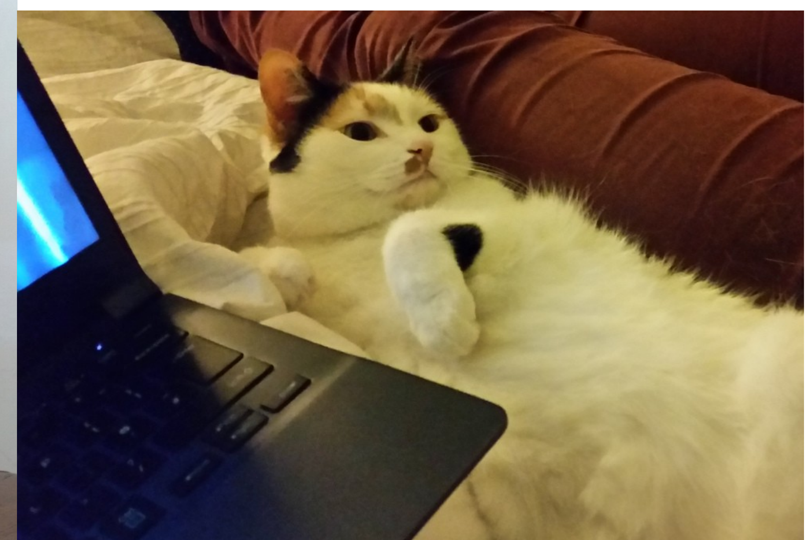
Synchronised multi-device media playback with GStreamer

Luis de Bethencourt
Samsung Open Source Group
luisbg@osg.samsung.com

About Me

- Originally from the Canary Islands. Currently in London.
- Joined GStreamer in 2010
- Working @ Samsung's Open Source Group



Agenda

- Motivation
- GStreamer is Pipelines
- GStreamer Clocks
- Setting up the Pipeline
- Examples

Motivation

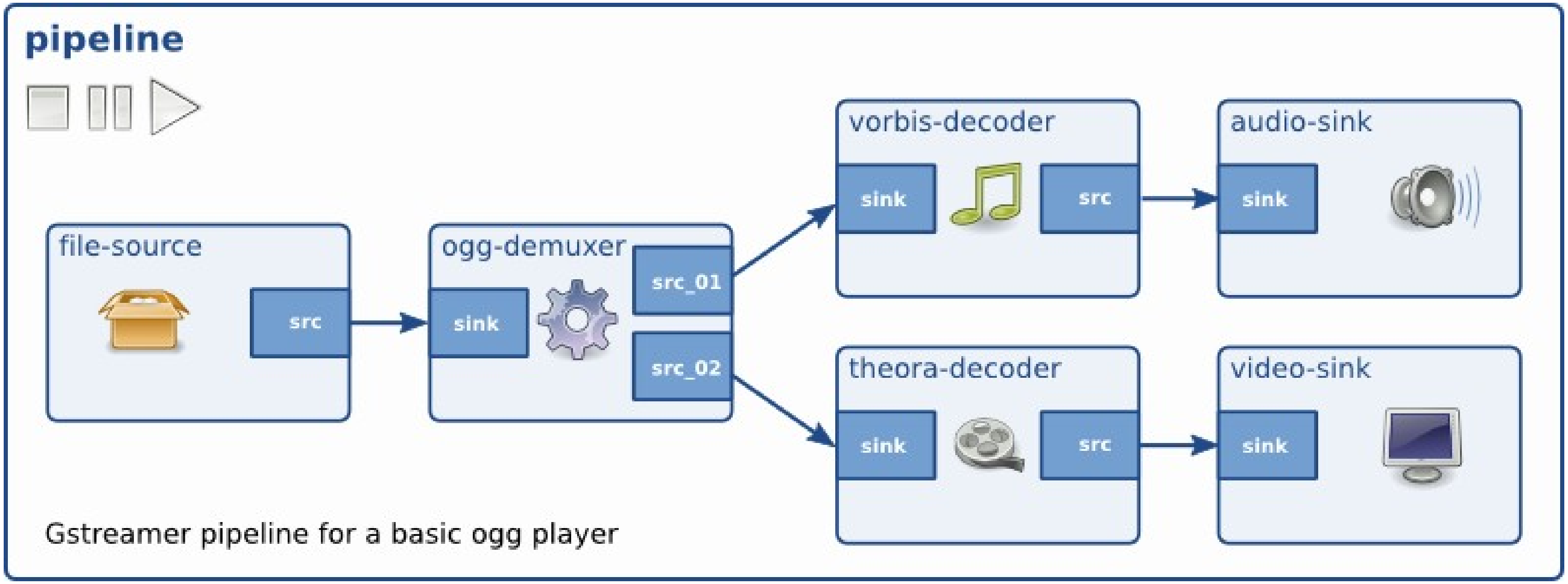
Motivation

- GStreamer is a large and global collaborative software development project
- Adding features like synchronised playback in your GST applications is easy
- Synchronised playback is useful
  - Media following you around the house
  - Mixing of live video streams
  - Video wall
  - Time based media analysis
- This talk will present how this works and how to use it

GStreamer is Pipelines

GStreamer is Pipelines

- GStreamer is a pipeline-based framework for creating media applications
- Pipeline = a set of data processing elements connected in series, where the output of one element is the input of the next one

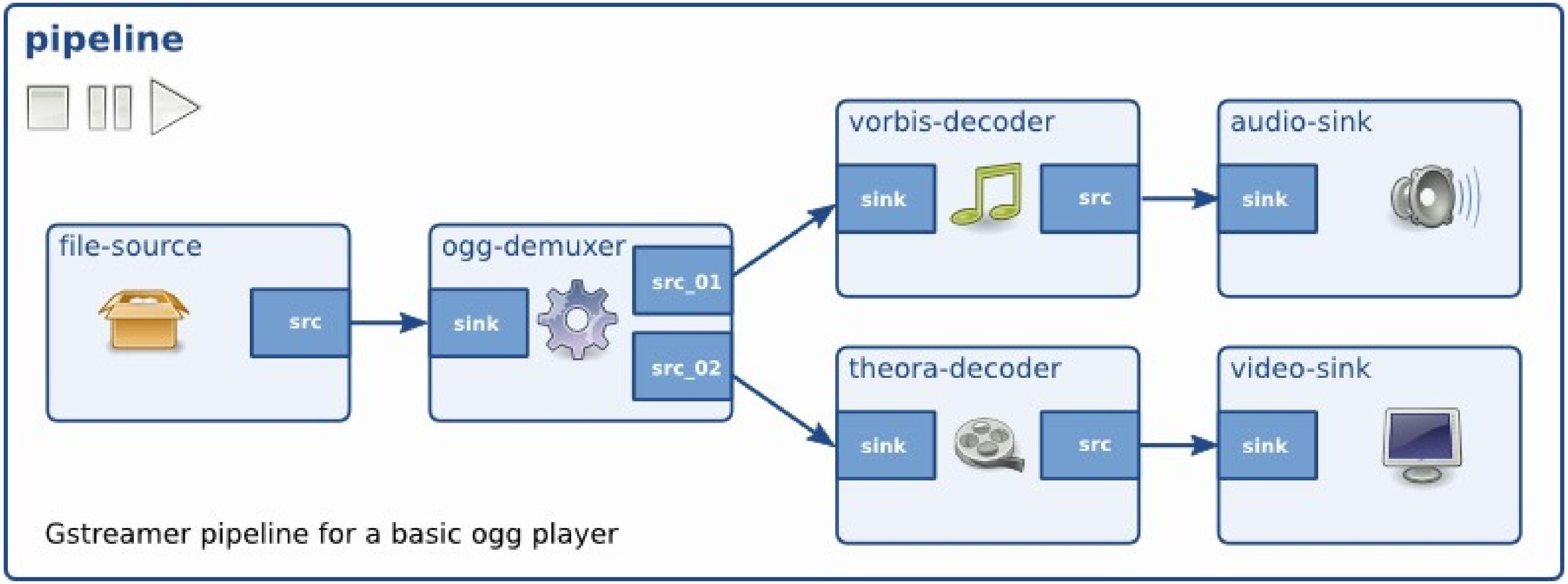


gst-launch filesrc location=”example.ogv” ! oggdemux name=”demux” ! vorbisdec ! autoaudiosink ! demux. ! theoradec ! autovideosink

GStreamer Clocks

GStreamer Clocks

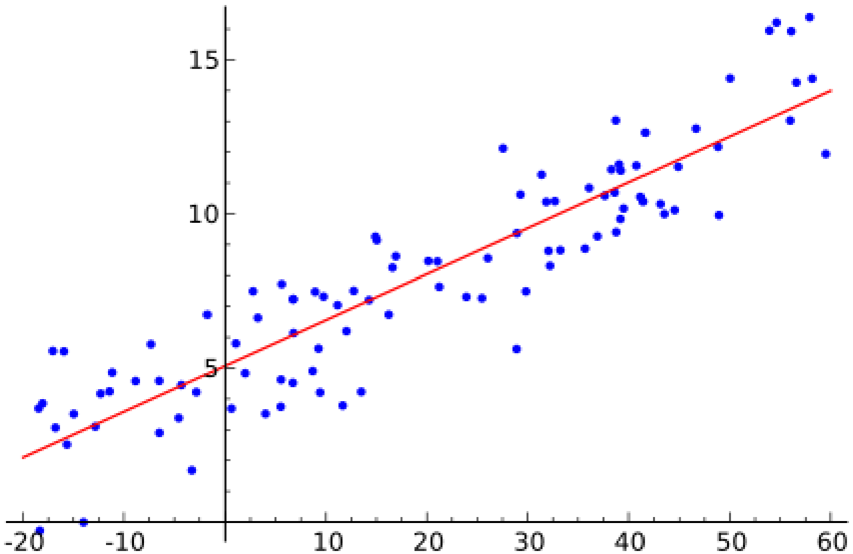
- A clock provider is an element in the pipeline that can provide a *GstClock* object
- The clock object needs to report an absolute-time that is monotonically increasing
- If an element with an internal clock needs to synchronize, it needs to estimate when a time according to the pipeline clock will take place according to the internal clock. To estimate this, it needs to slave its clock to the pipeline clock



Gstreamer pipeline for a basic ogg player

GStreamer Clocks

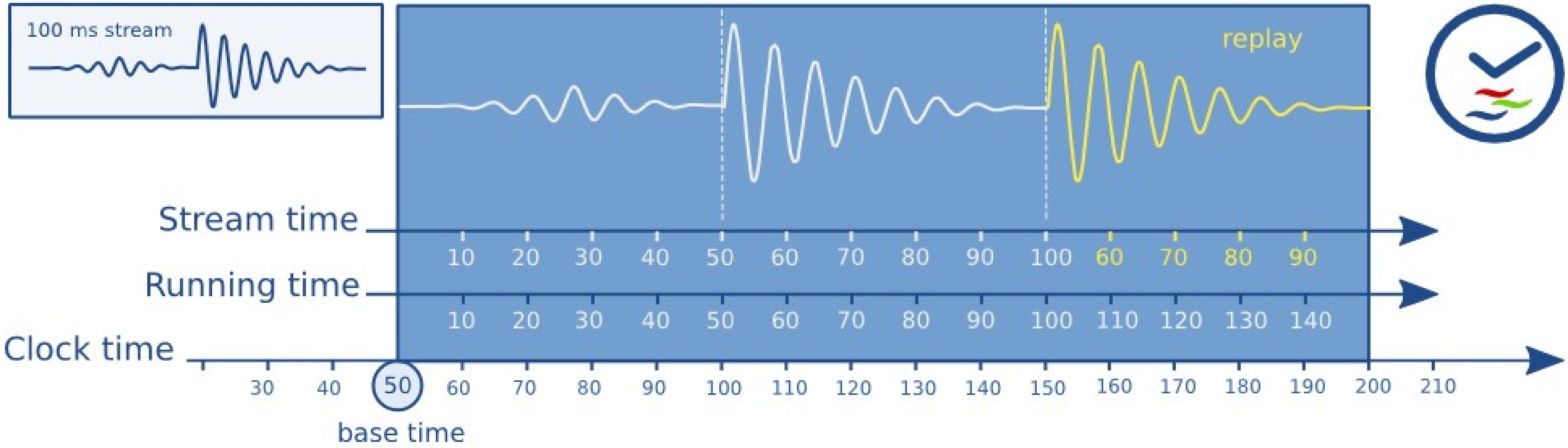
- Types of clock slaving:
  - Skew
    - This is the default method. Compares the drift between internal and the master clock and compensates when it exceeds a maximum allowed drift.
  - Resample
    - Does observations on the master clock and uses linear regression to adjust the base and offset used by the internal clock.



GStreamer Times

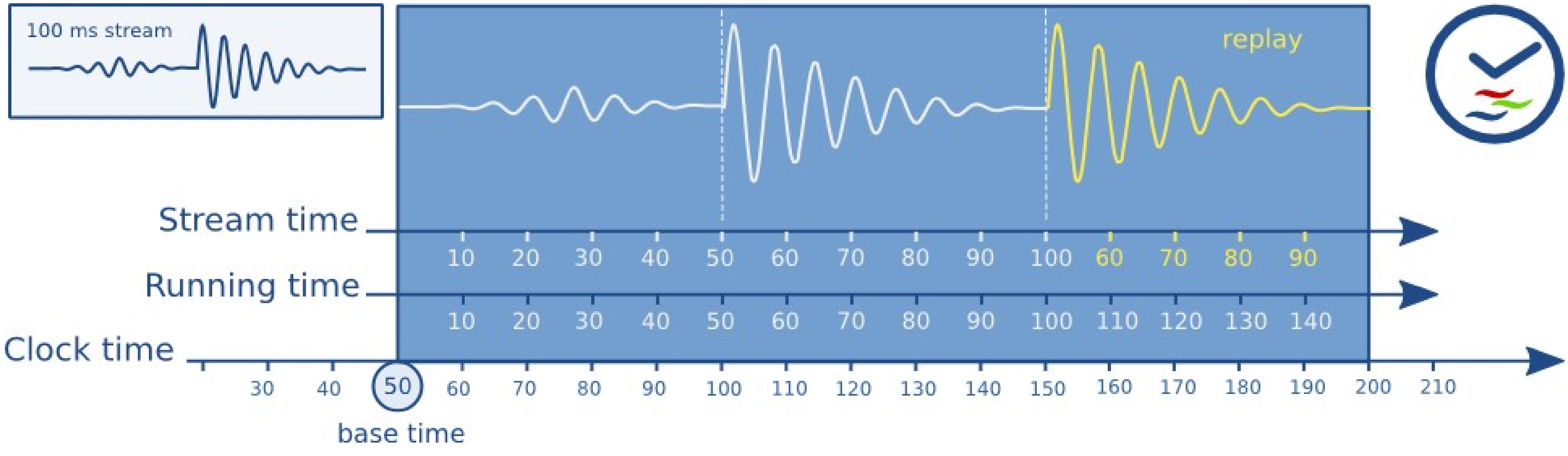
- A GstClock returns the absolute-time with *gst_clock_get_time*()
- base-time is the absolute-time when it changed to PLAYING state
- running-time is the total time spent in the PLAYING state
- running-time = absolute-time – base-time

times in the pipeline when playing a 100ms sample and repeating the part between 50ms and 100ms.



GStreamer Times

- stream-time represents the time inside the media as a value between 0 and the total duration of the media. Used for position and seeks
- Synchronization is now a matter of making sure that a buffer with a certain running-time is played when the clock reaches the same running-time
- Usually this task is done by sink elements



GStreamer NetClocks

- For synchronising devices we use more than one clock
- No two clocks show the same time
- No two clocks run at the same rate
- We need a way to approximate the same time on multiple devices

- **Solution**: using the *GstClock* class,
  - create a clock that bases it's internal time on another machine in the network
  - slave it to the local system clock

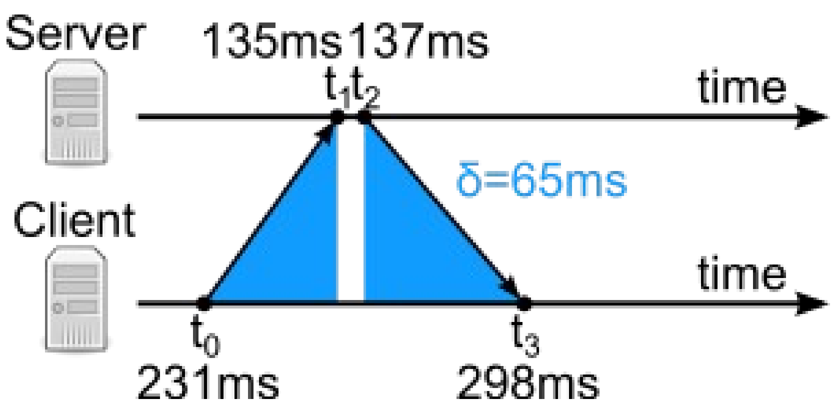
GStreamer NetClocks

- *GstNetClientClock* since ~2005
  - Custom protocol
  - *gst_net_client_clock_new*()
- *GstNtpClock* (NTPv4) since 1.6 release (June 2015)
  - Shares most of the code with GstNetClock
  - *gst_ntp_clock_new*()
- GstPtpClock (IEEE1588:2008) since 1.6 release (May 2015)
  - Higher accuracy in local system (ns compared to NTP's ms)
  - Possibility of network hardware support which increases accuracy
  - Less robust in networks with fluctuating RTTs (eg, WiFi)
  - *gst_ptp_init*(); *gst_ptp_clock_new*()

GStreamer NetClocks

- Complexity lies on handling the Round-Trip delay time

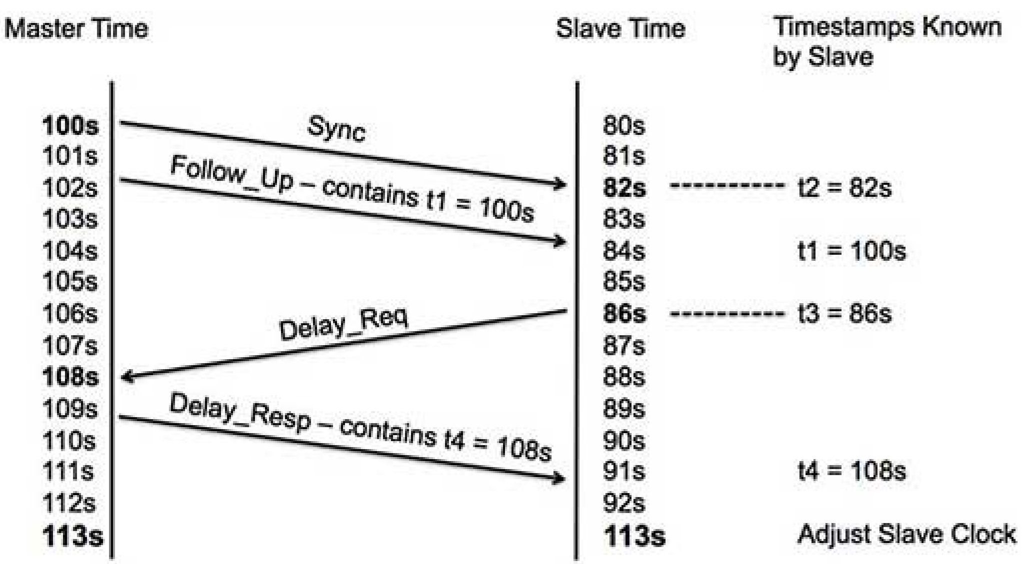
NTPv4



$$\theta = \frac{(t_1 - t_0) + (t_2 - t_3)}{2}$$

$$\delta = (t_3 - t_0) - (t_2 - t_1)$$

PTP



Media Transport

Media Transport

- All devices need to have access to the same media
- Possible choices:
  - HTTP
    - Easier to do buffering
    - No worries about firewalls
  - DASH/HLS
    - Good CDN usage
    - Multiple bitrates/resolutions
  - RTP/RTSP
    - The most “automatic”
    - Great for low-latency streaming

Setting up the Pipeline

Setting up the pipeline

- *gst_pipeline_use_clock*()
  - Forces the usage of a specific clock
  - Set the same network clock on all devices
- *gst_element_set_base_time*();
  - Matches the running time all devices to the same absolute-time
- *gst_element_set_start_time*();
  - Disable the distribution of the base_time to the children
- *gst_pipeline_set_latency*()
  - Overrides default pipeline latency handling to use static latency
  - Should be at least the maximum receiver latency
    (network + decoder + latency)

Examples

playbin

```
gst_init (&argc, &argv);

/* Create the element */
playbin = gst_element_factory_make ("playbin", "playbin");
g_object_set (playbin, "uri", "http:///just/an/example", NULL);

client_clock = gst_net_client_clock_new (NULL, "192.168.1.42", clock_port, 0);
base_time = get_base_time ();

/* Set up synchronisation */
gst_pipeline_use_clock (GST_PIPELINE (playbin), client_clock);
gst_element_set_start_time (playbin, GST_CLOCK_TIME_NONE);
gst_element_set_base_time (playbin, base_time);

/* Play */
gst_element_set_state (playbin, GST_STATE_PLAYING);
```

Demo

Sample code at:
https://github.com/luisbg/synchronised_media_playback

gst-rtsp-server

- Examples in gst-rtsp-server/examples:
  - http://cgit.freedesktop.org/gstreamer/gst-rtsp-server/tree/examples/
- test-netclock
  - Sets up netclock provider
  - Uses system's clock for pipeline and netclock provider
- test-netclock-client
  - Sets up netclient's clock with sender's server
  - Use that for pipeline clock and set fixed latency to 500ms

Aurena

- https://github.com/thaytan/aurena
- It provides:
  - a media content server
  - a client for synchronised playback across all receivers
- Clients autodiscover the server via Avahi
- Controlled through web interface in server

Questions?

Find Me

- If you have any questions or wanted to learn anything else Gstreamer or Samsung Open Source Group related…

- luis@debethencourt.com
  luisbg@osg.samsung.com
  luisbg @ freenode
  @luisbg

Thank You!

Slides will be shared soon at:
http://www.slideshare.net/SamsungOSG