Debugging race condition problems in GStreamer

gstreamer Conference 2016

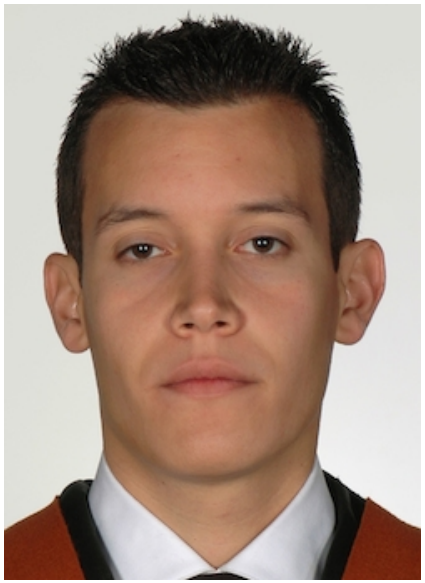
10-11 October 2016
Berlin, Germany

Miguel París
mparisdiaz@gmail.com

Who I am

Miguel París

- Software Engineer
- Telematic Systems Master's
- Kurento real-time responsible
- mparisdiaz@gmail.com
- Twitter: @mparisdiaz



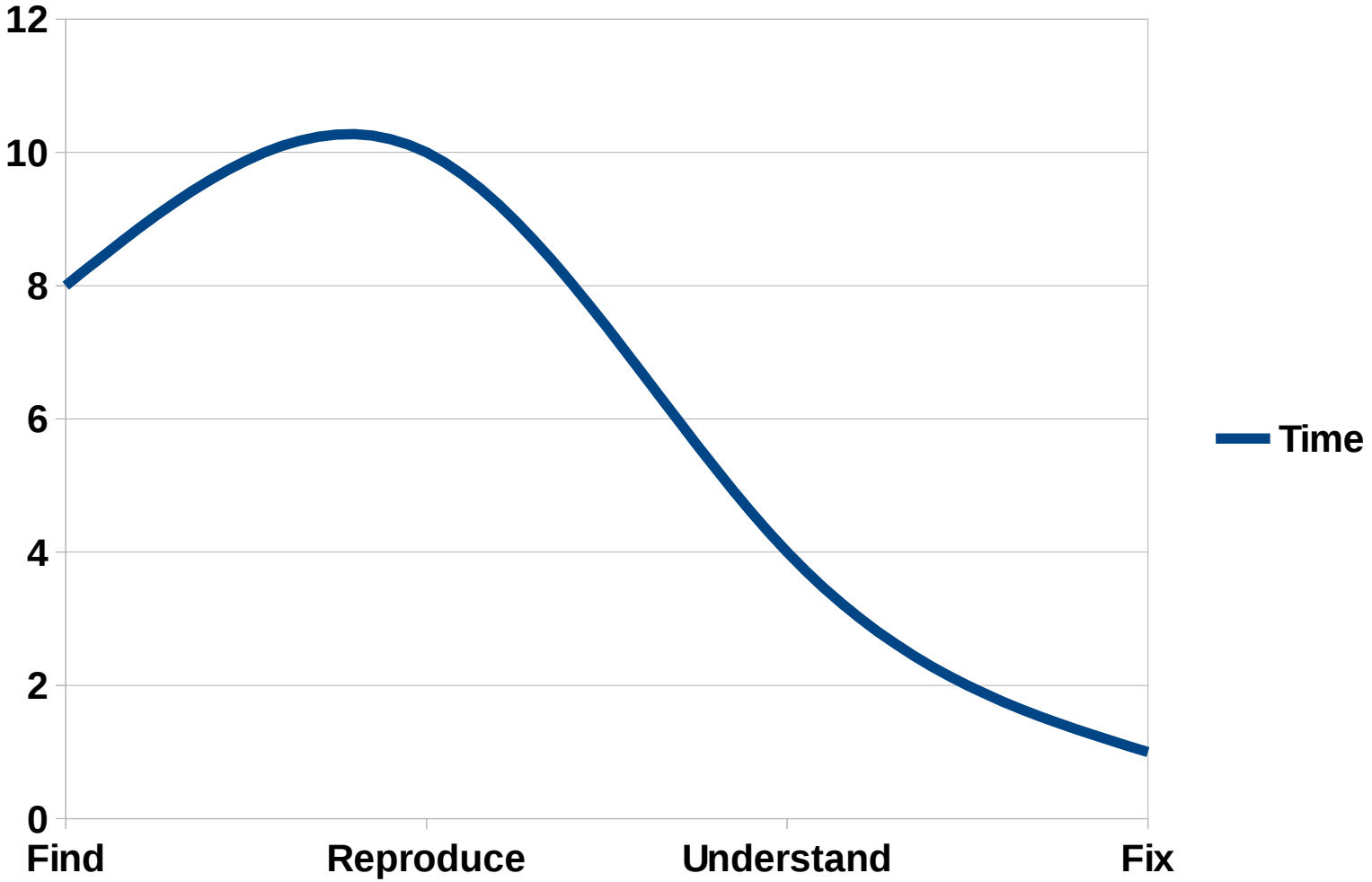
Overview

- Dealing with multi-threaded systems is not easy in general
- Systems related to media with real-time restrictions are even much more complicated
  - Critical bugs are only seen under specific race conditions that only take place time to time
  - Debugging is a hard work that can consume a lot of time
- Some bugs found in the “Kurento context” due to we work with dynamic pipelines (performing changes while pipeline is running)
  - Link
  - Unlink
  - Renegotiate

Process

1) Find
   - Production
   - Tests
2) Reproduce
   - Can we create a simple test/program to reproduce the problem?
   - It is not needed that it always fails
3) Understand
   - Can we gather enough info to understand the problem?
   - Use the info to develop a better test/program applying hacks to force race conditions
4) Fix
   - Check that the problem does not take place
   - Deeply think about possible drawbacks of the fix

Time consumption



Tool set

- Bug finders
  - forever.sh (run until failure)
- Bug hunters
  - GDB
  - valgrind
  - G_DEBUG=fatal_warnings
  - Specific logs
- Race condition provokers
  - sleep
  - cond_wait/cond_signal

Using tools

- How can we find bugs if the race conditions only happen time to time (e.g.: 1/1000)?
  - Use “bug finders”
    - No problem, run as many times you need (e.g.:1000 times)
    - Automatic and background way: do not spend developer time
    - Much better if it can be reproduced by an automatic test
- How can we gather the info when the bug happens?
  - Use “bug hunters”
    - Then analyze outputs and reports
- How can we make a test/program that fails almost always?
  - Use “race condition provoker”
  - This will help you to understand the problem

Detected Race Condition Bugs

- tee: Avoid race condition while forwarding sticky events
  - https://bugzilla.gnome.org/show_bug.cgi?id=752213
- tee: adding inactive pad to running element
  - https://bugzilla.gnome.org/show_bug.cgi?id=772115
- pad: check caps not NULL before referring
  - https://bugzilla.gnome.org/show_bug.cgi?id=768450
- ghostpad: invalid ref getting internal pad
  - https://bugzilla.gnome.org/show_bug.cgi?id=768100
- gstclock: segmentation fault when unschedule
  - https://bugzilla.gnome.org/show_bug.cgi?id=770953

Analyzing a real case (I)

- The **goal is not** that the audience deeply understand the case, but see how much complicated this kind of bugs can be and how to apply the process.
- tee: Avoid race condition while forwarding sticky events
  - https://bugzilla.gnome.org/show_bug.cgi?id=752213
  - Critical warnings related to **tee** and **pad** found in some Kurento tests:

```
Unexpected critical/warning:
gstpad.c:4258:gst_pad_push_data:<tee0:src_1> Got data flow
before segment event

GStreamer-WARNING **:
gstpad.c:5031:store_sticky_event:<tee0:src_1> Sticky event
misordering, got 'caps' before 'stream-start

GStreamer-WARNING **:
gstpad.c:5059:store_sticky_event:<fakesink1:sink> Sticky
event misordering, got 'caps' before 'stream-start'
```

Analyzing a real case (I)

- tee: Avoid race condition while forwarding sticky events
  - https://bugzilla.gnome.org/show_bug.cgi?id=752213
  - Critical warnings related to **tee** and **pad** found in some Kurento tests:

```
Unexpected critical/warning:
gstpad.c:4258:gst_pad_push_data:<tee0:src_1> Got data flow
before segment event

GStreamer-WARNING **:
gstpad.c:5031:store_sticky_event:<tee0:src_1> Sticky event
misordering, got 'caps' before 'stream-start

GStreamer-WARNING **:
gstpad.c:5059:store_sticky_event:<fakesink1:sink> Sticky
event misordering, got 'caps' before 'stream-start'
```

Analyzing real case (II)

- Set environment to “hunt” the error

```
$> echo "core" |sudo tee
/proc/sys/kernel/core_pattern

$> ulimit -c unlimited

$> G_DEBUG=fatal_warnings ./forever.sh run.sh
```

- Ref: man core
  - http://man7.org/linux/man-pages/man5/core.5.html

Analyzing a real case (III)

[app_thread] **[streaming_thread]**

1 - tee0 and fakesink0 are linked

2 - stream-start event arrives to the **tee0:sink** pad

2.1 - it is forwarded to **tee0:src_0** and **fakesink0:sink**

3 - Just:

- after forwarding the event to all tee src pads
- and before storing the sticky event in tee0:sink pad

a new tee src pad is added (**tee:src_1**) and linked to **fakesink1:sink**

- The stream-start is NOT forwarded to tee:src_1 because the forwarding iteration has already finished
- the stream start is NOT stored in tee:src_1 because tee0:sink has not stored the event yet

4 - caps event arrives to the tee0:sink pad

4.1 - it is forwarded to all tee src pads and to fakesink0:sink and fakesink:1:sink pads

So, fakesink1:sink receives the caps event without having the stream-start event

5 - Performs

5.1 - fakesink1:sink is unlinked from **tee:src_1**

5.2 - tee:src_1 is released

5.3 - fakesink1:sink is linked to a new tee src pad (**tee:src_2**)

5.3.1 - stream-start event is stored in tee:src_2

5.3.2 - stream-start event is tried to be stored into fakesik1:sink

Here we have the misordering error

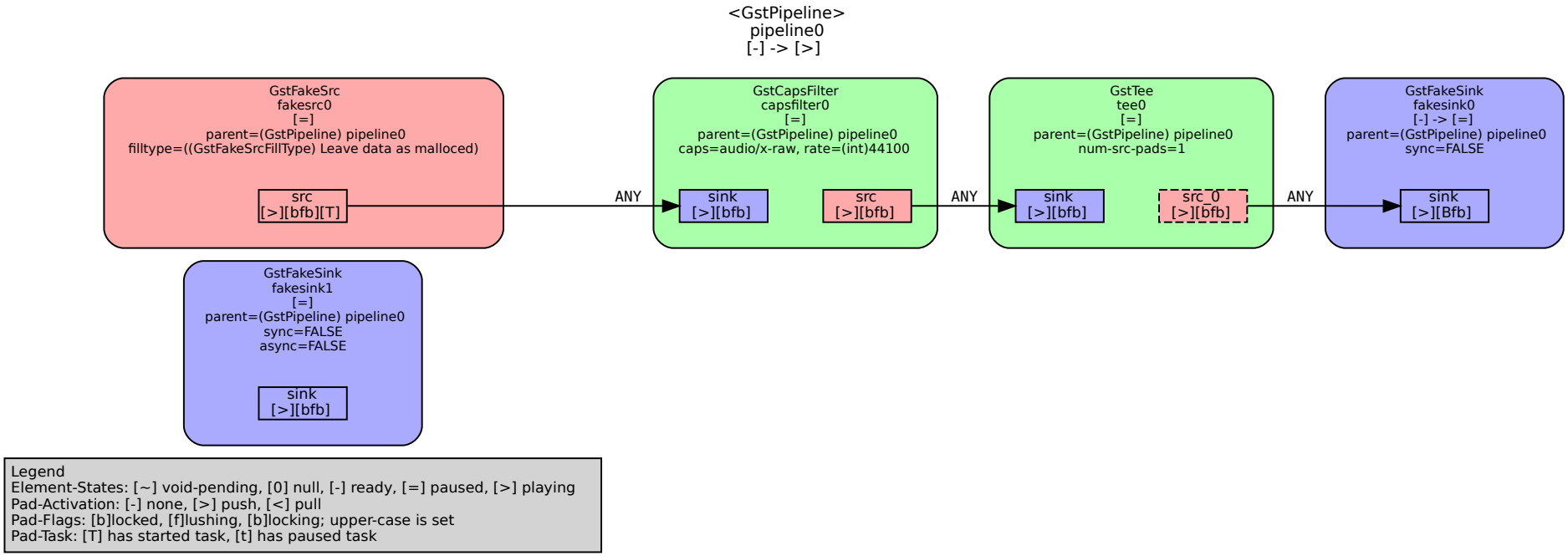
Analyzing real case (IV)

[app_thread]
[streaming_thread]

1 - tee0 and fakesink0 are linked

2 - stream-start event arrives to the **tee0:sink** pad

2.1 - it is forwarded to **tee0:src_0** and **fakesink0:sink**



Analyzing real case (V)

[app_thread] **[streaming_thread]**

3 - Just:

- after forwarding the event to all tee src pads
- and before storing the sticky event in tee0:sink pad

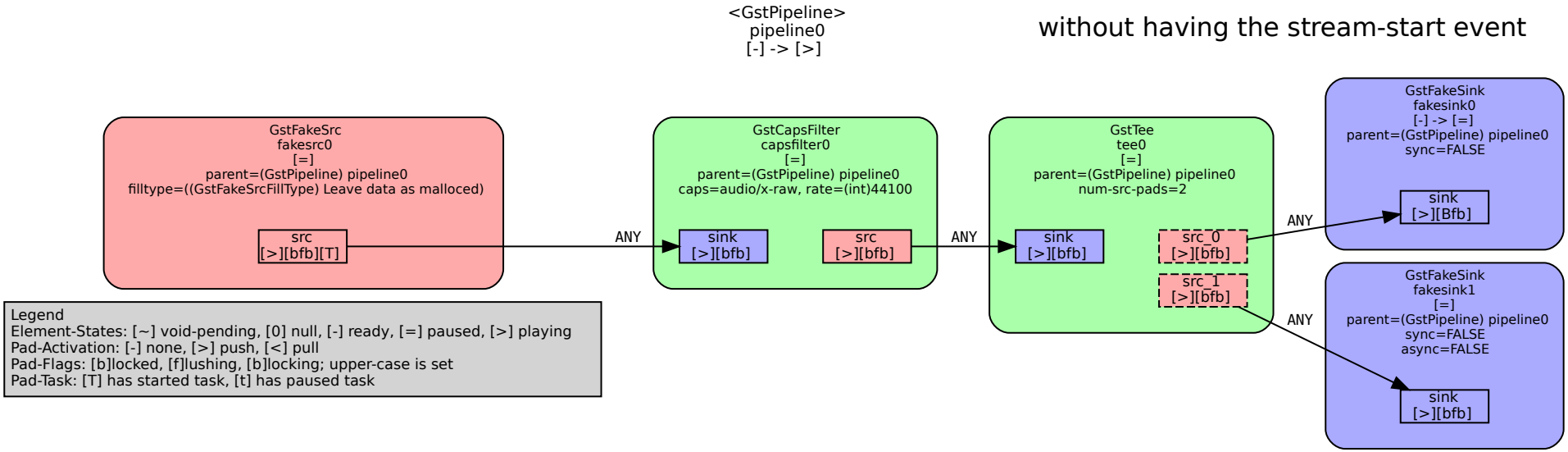
a new tee src pad is added (**tee:src_1**) and linked to **fakesink1:sink**

- The stream-start is NOT forwarded to tee:src_1 because the forwarding iteration has already finished
- the stream start is NOT stored in tee:src_1 because tee0:sink has not stored the event yet

4 - caps event arrives to the tee0:sink pad

4.1 - it is forwarded to all tee src pads and to fakesink0:sink and fakesink:1:sink pads

So, **fakesink1:sink** receives the caps event without having the stream-start event



Analyzing real case (VI)

[app_thread] **[streaming_thread]**

5 - Performs

5.1 - fakesink1:sink is unlinked from **tee:src_1**

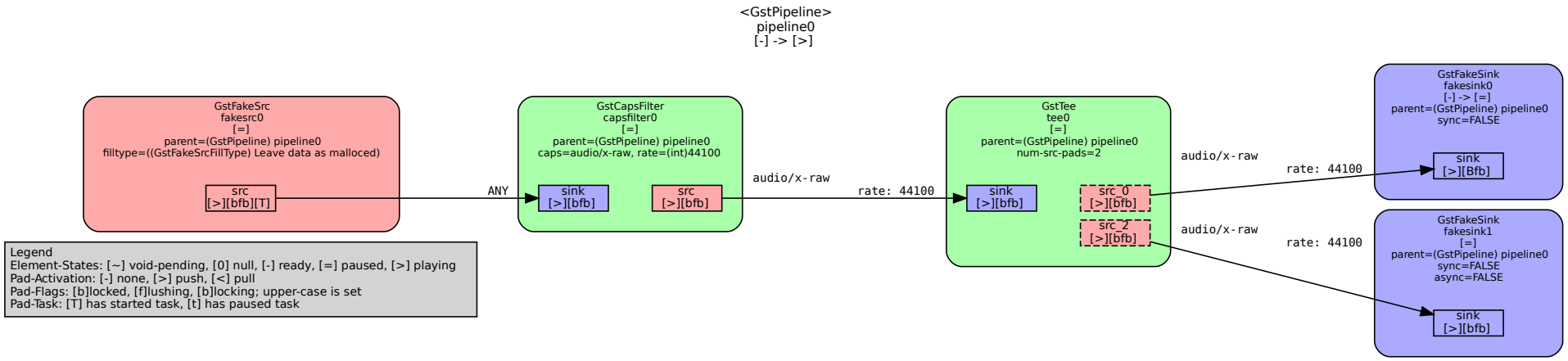
5.2 - tee:src_1 is released

5.3 - fakesink1:sink is linked to a new tee src pad (**tee:src_2**)

5.3.1 - stream-start event is stored in tee:src_2

5.3.2 - stream-start event is tried to be stored into fakesik1:sink

Here we have the misordering error



General remarks

- Invest some minutes to think about race conditions when you are developing. In this way you can save days (even weeks) debugging when the bug appears
  - For that you can use this idea: “putting sleeps in the code should work like without them”
- Deadlocks are easier to debug that “open critical sections”
  - GStreamer has a lot of “open critical section” to avoid deadlock due to use mutex instead of recursive mutex
- Use g_warning/g_critical when you consider that the situation is wrong
  - It is better being quite strict with that and add too g_warnings and remove they later than do not detect wrong situations

Conclusions/Future work

- Debugging race conditions problem can consume a lot of time
  - Automate fully or partially the process → Continuous Integration
  - How?
    - Use free slots of the nightlies to run forever.sh of some tests
    - Use “bug hunters” to gather info if a bug happens
- Tests only cover part of the system
  - What can we do?
    - Stress your systems looking for bugs
    - Use maintenance periods to use “race condition provokers” to look for bugs

Thank you

http://www.kurento.org
http://www.github.com/kurento
info@kurento.org
Twitter: @kurentoms

http://www.nubomedia.eu

http://www.fi-ware.org

http://ec.europa.eu

Miguel París
mparisdiaz@gmail.com