GStreamer Element States

How do they work in detail?

GStreamer Conference 2016, Berlin

10 October 2016

Sebastian Dröge <sebastian@centricular.com>

Introduction

Who?

- Long-term GStreamer core developer and maintainer since 2006
- Did the last few GStreamer releases and probably touched every piece of code by now

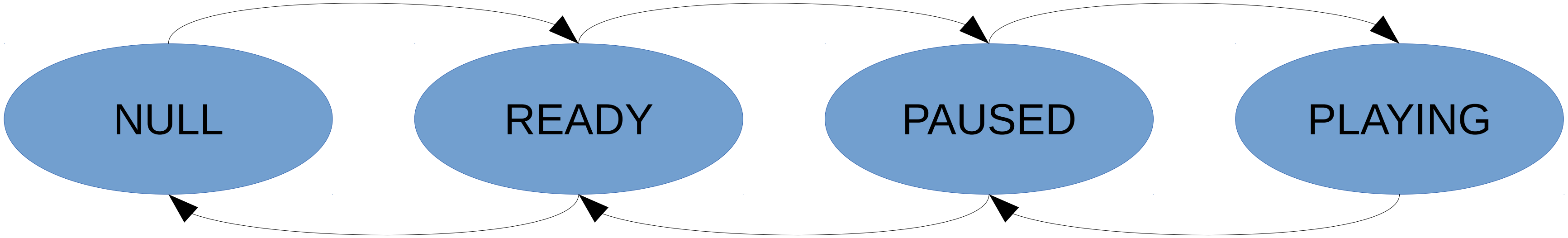
- One of the founders of Centricular Ltd
  - Consultancy offering services around GStreamer, graphics and multimedia related software

What?

- How do GStreamer element states work?
- Internals you usually don’t have to worry about
- Problems with the current design
- Ideas for a better future

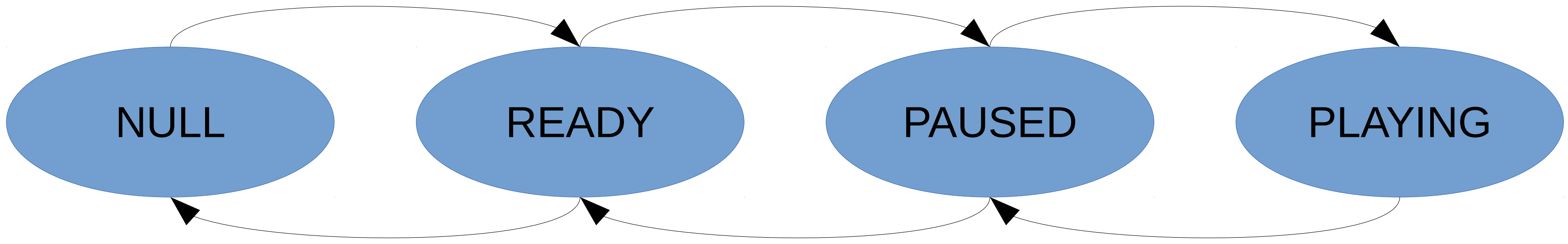
The States

State Transitions



State Transitions (2)

- NULL: Deactivated, element occupies no resources

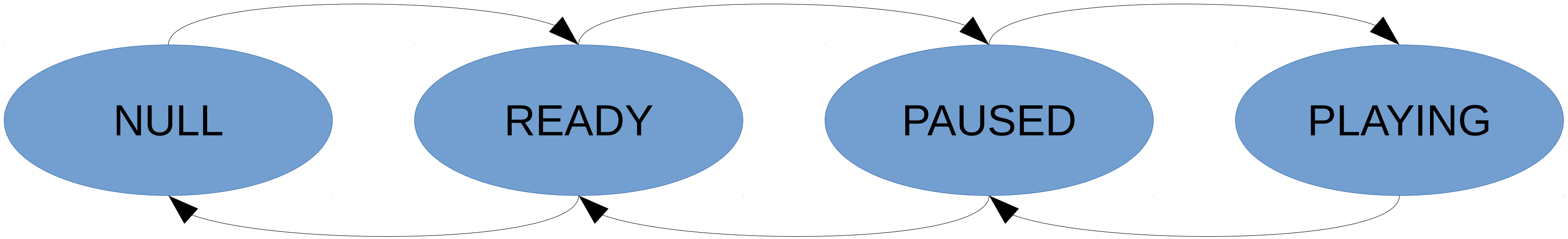


gstreamer

Centricular

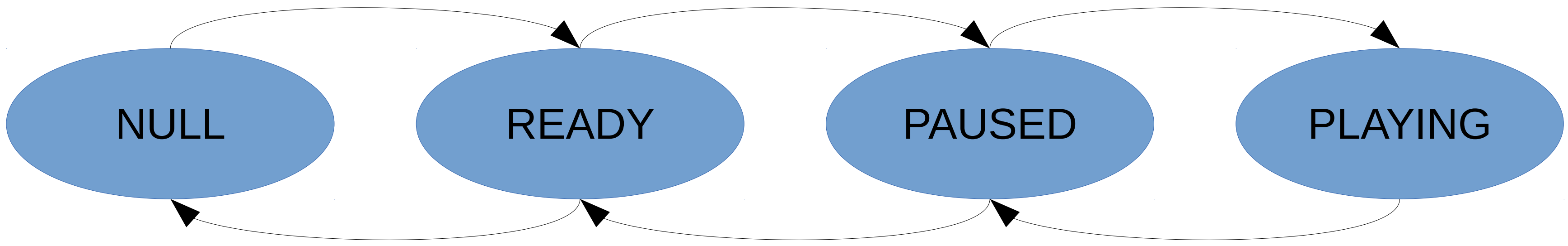
State Transitions (3)

- READY: Check and allocate resources



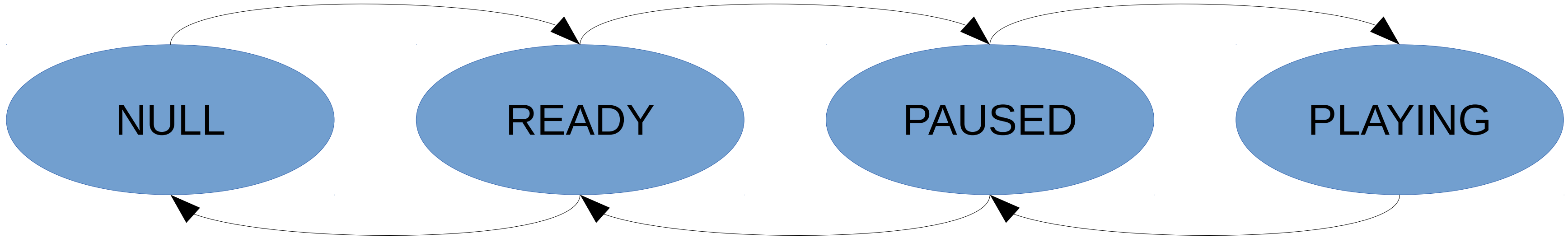
State Transitions (4)

- PAUSED: pre-roll, i.e. get a buffer to each sink



State Transitions (5)

- PLAYING: active dataflow, running-time is increasing



State Changes

Let’s start simple

Short Overview of Relevant API

Application API

- `gst_element_set_state(element, state)`
  - `Returns: SUCCESS/FAILURE/others later`
- `gst_element_get_state(element, *state, *pending, tout)`

- `STATE_CHANGED(old, new, pending) message`
- `ERROR message` – aborts state changes
- More messages later

Virtual Methods

- **`change_state(element, transition):`**
  `Do what is needed to change the state`

Usually never used outside GstPipeline/Bin/Element itself

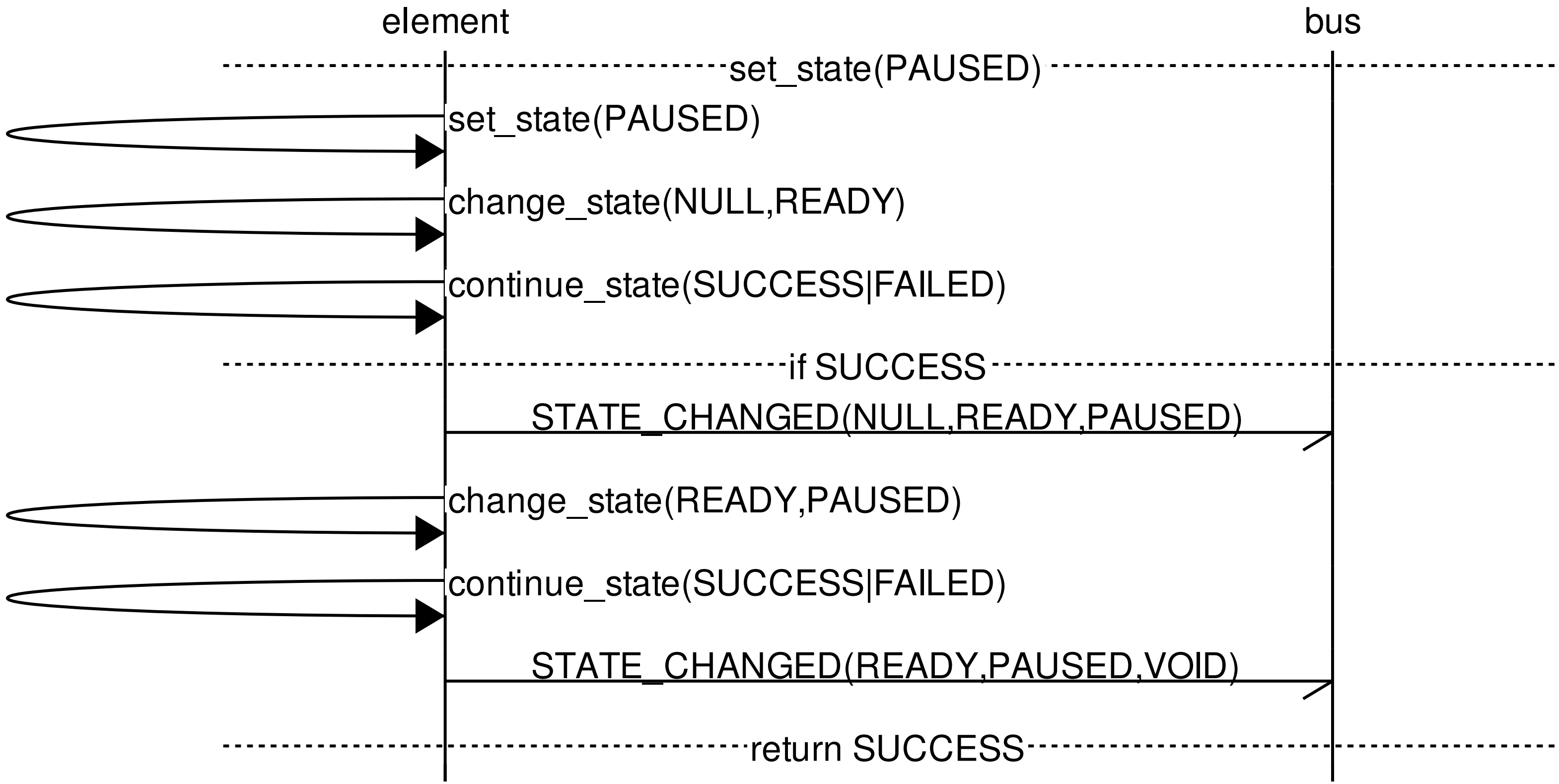
- **`state_changed(element, old, new, pending):`** `Notification`
- **`set_state(element, state)`**
- **`get_state(element, state, pending, timeout)`**

Internal State Tracking

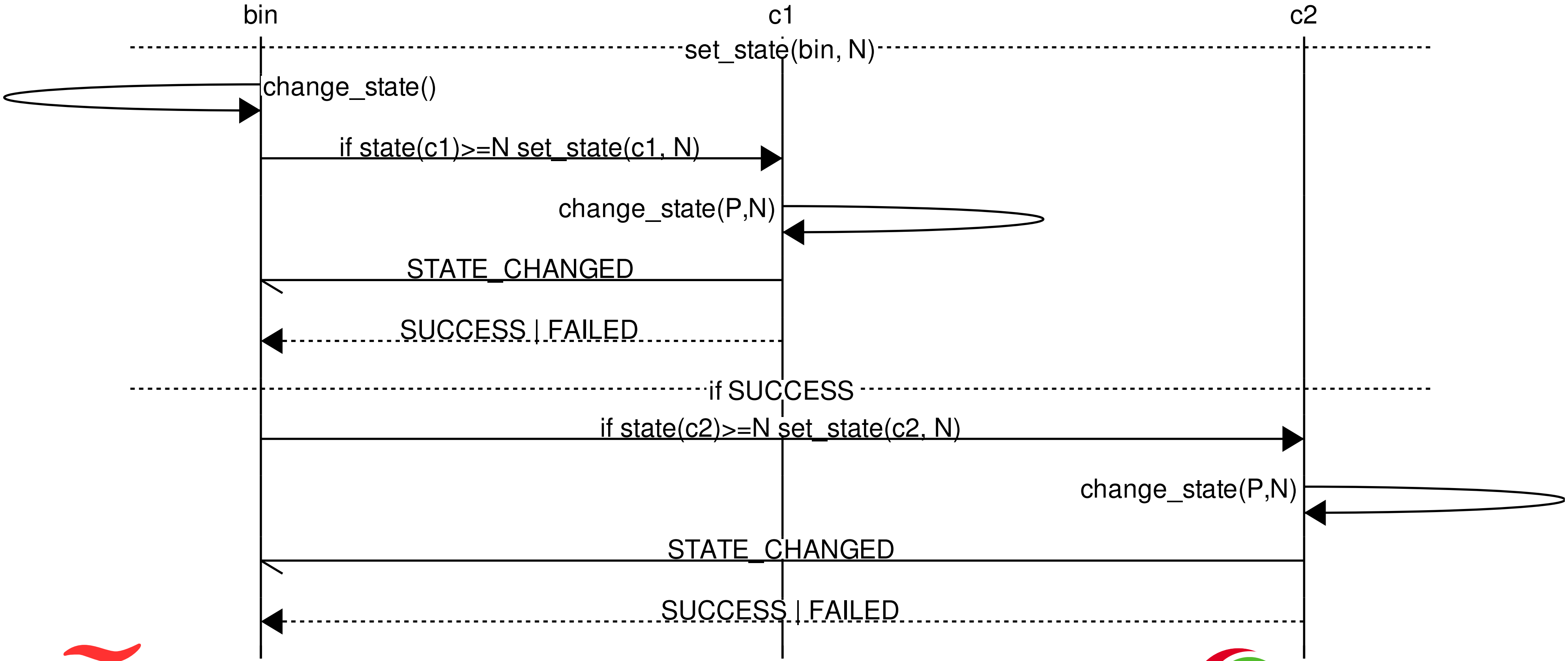
- **`target_state`:** Set by set_state(), final target
- **`current_state`:** Where are we now?
- **`next_state`:** What state are we going to right now?
- **`pending_state`:** What is the intermediate final state? Later!

- **`last_return`:** Keep track of last state success/failure
- **`state_{lock,cond,cookie}`:** Locking / detection of concurrent change

Element – Intermediate States States



GstBin – Manages Child States



GstBin – Traps & Quirks

- Not if element is added/removed
  - `set_state()`
  - `sync_state_with_parent()`: Sets parent's **pending** state

- Not if `gst_element_set_locked_state(child, TRUE)`

gstreamer

Centricular

GstPipeline – Makes it all work together

- PAUSED → PLAYING
  - Select a clock
  - Measure & set base time
- PLAYING → PAUSED
  - Measure start time, how much time was spent in PAUSED?

- Go watch Nicolas' talk tomorrow for what that means!

All Synchronous? No!

- Acquiring resources can block
  - Network, hardware, …

- Welcome to asynchronous state changes

Asynchronous State Changes

If you don’t want to wait

Single Element

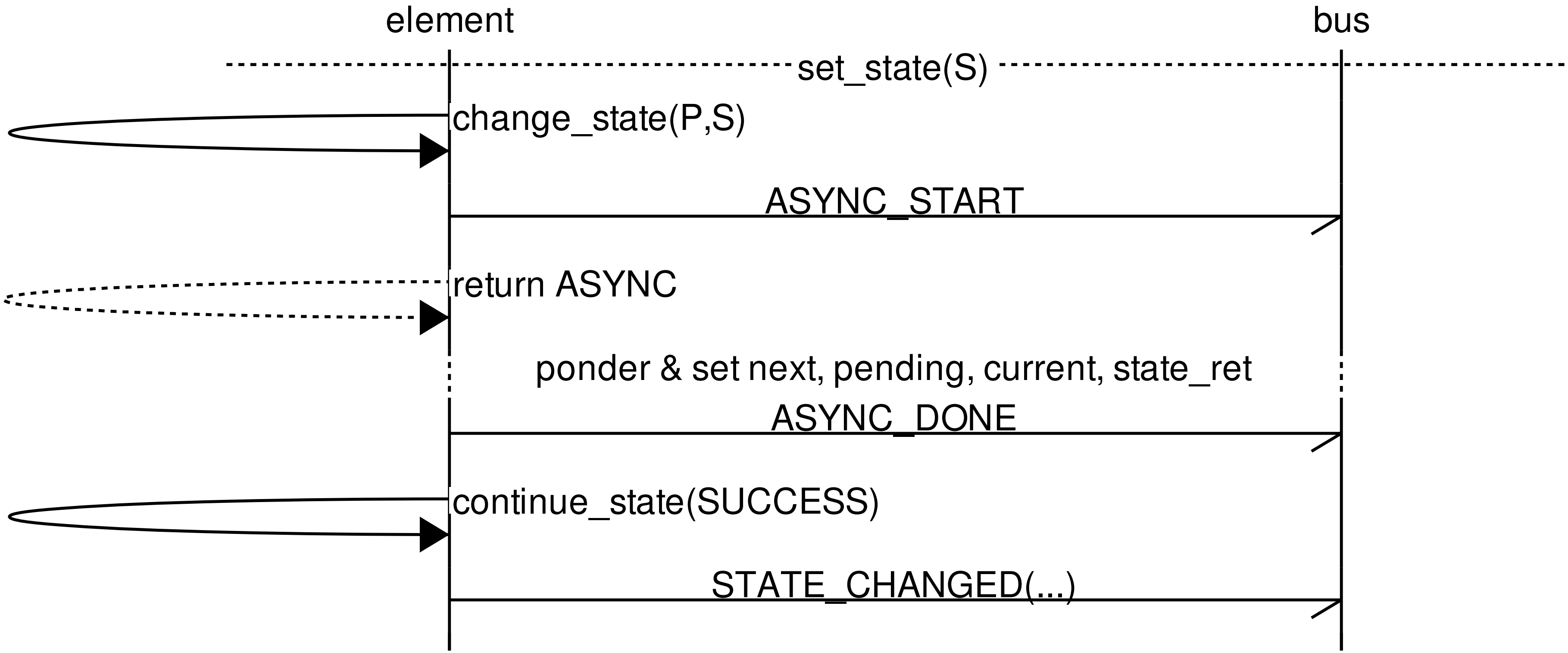
- Starts easy!

- `change_state()` returns `ASYNC`
- `ASYNC_START / ASYNC_DONE` messages

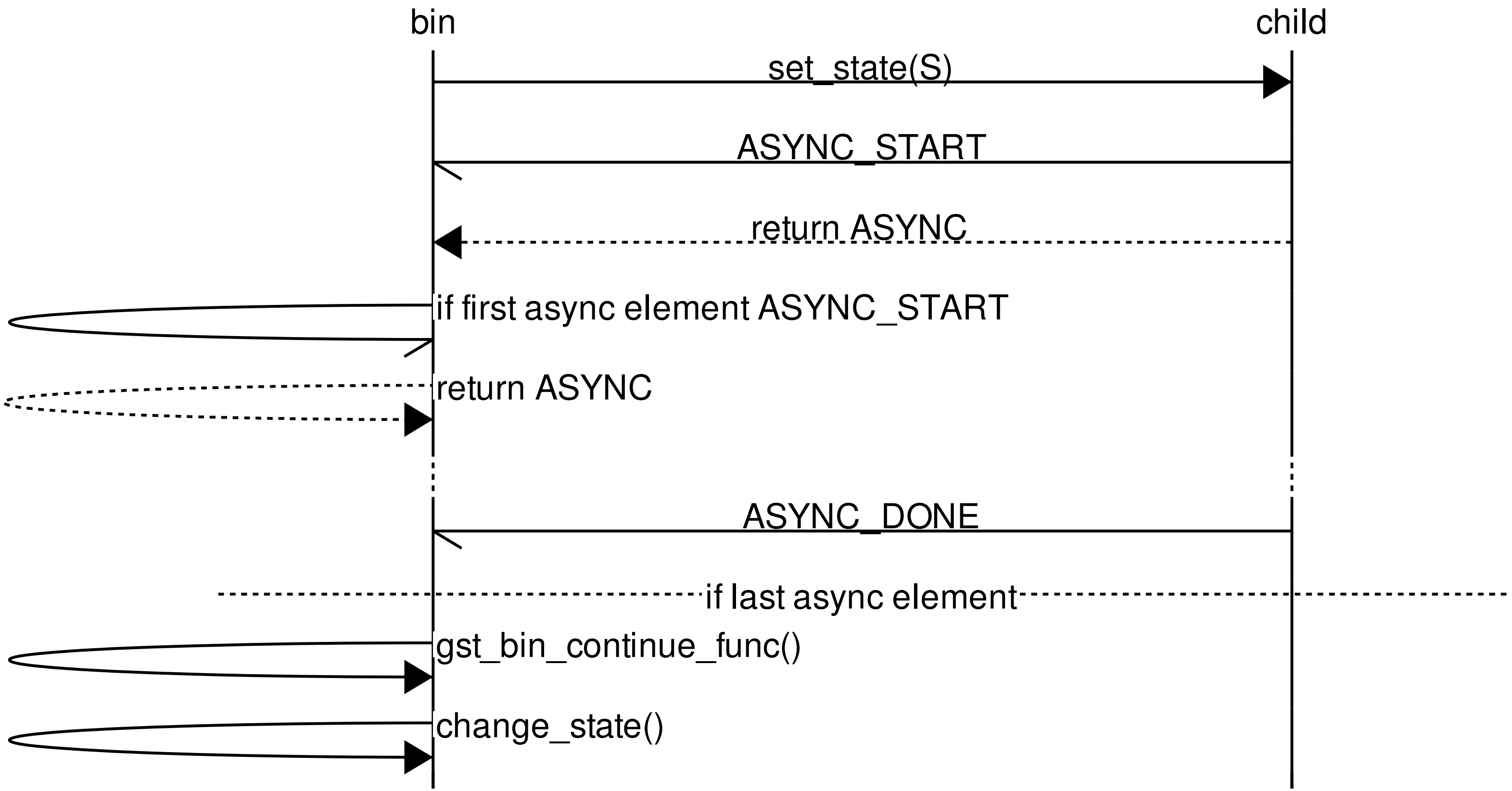
Single Element (2)

- When done, post `ASYNC_DONE` and
  - `gst_element_abort_state() & ERROR` message on failure, or
  - `gst_element_continue_state()*`, or
  - GstBin subclass: `GstBin::handle_message()` instead of posting directly

Single Element (3)



GstBin – State Tracking



GstBin – State Tracking

- Adding / removing elements checks for `ASYNC`
  - Can trigger new async state changes!
  - Can trigger PAUSED → PAUSED and similar
- async-handling property=true || top-level bin
  - Only these are doing the continue part
  - Only others are posting `ASYNC_START`

Progress Feedback

- `PROGRESS` messages – Only informative
- Start/Continue/Complete/Cancelled/Error
- String code & human readable text

- Used only by rtspsrc / rtspclientsink so far

Losing State

Async – Just the other way around

Elements Losing State

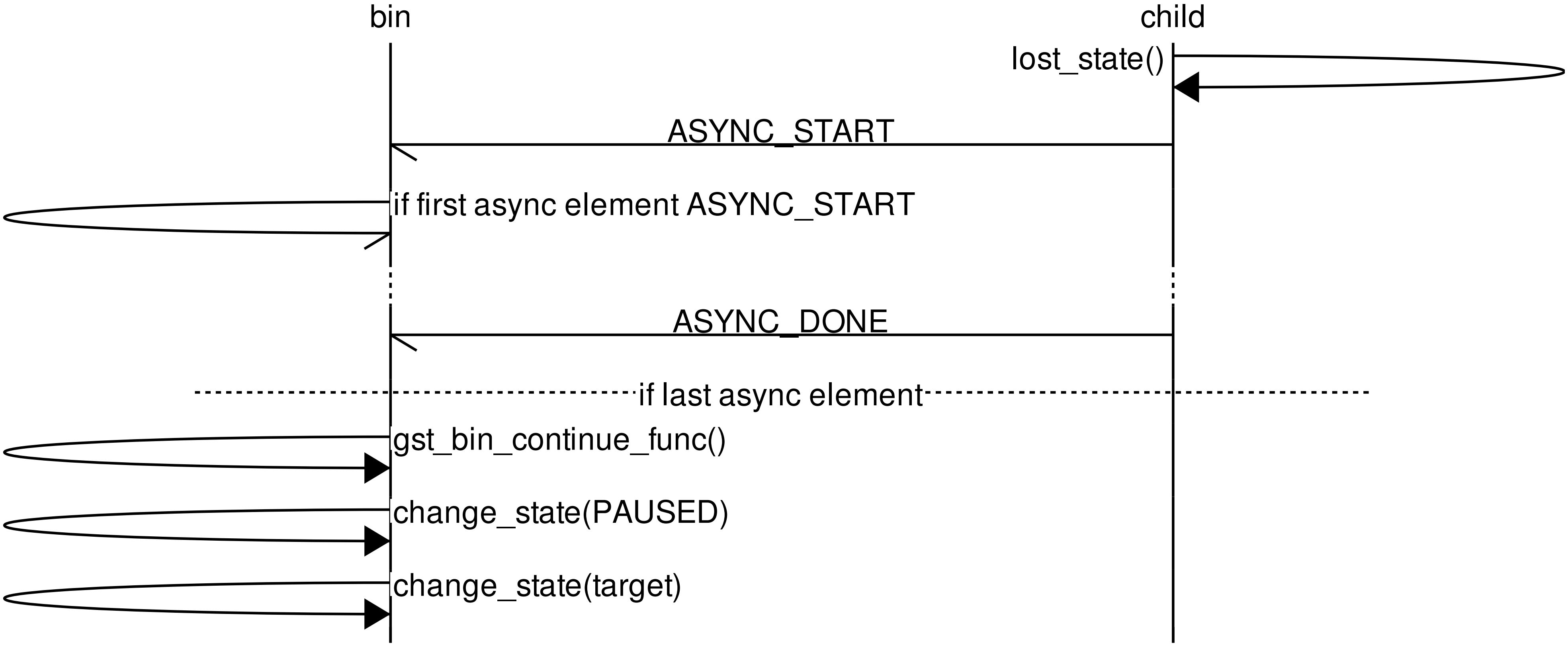
- Flushes (e.g. seek) move sinks out of PLAYING/PAUSED
  - `gst_element_lost_state()`
  - Posts `ASYNC_START, pending=PAUSED, target=old state, current=PAUSED, next=PAUSED`
  - Does **not** go through `change_state()`

- Triggered from an element inside the pipeline at any point in time, instead of being triggered from the outside!
- Handled in bins exactly the same

gstreamer

Centricular

Elements Losing State (2)



Live Elements

Don’t affect state changes… or do they?

NO_PREROLL

- change_state() returns neither SUCCESS nor ASYNC
  - NO_PREROLL!
- Considered like SUCCESS, overrides ASYNC

- Means
  - Don't stay in PAUSED for long
  - PAUSED is reached immediately (local) without waiting for preroll

gstreamer

Centricular

GstBin Handling

- Ignore all async elements and commit state immediately
- Ignore ASYNC_START
- Async elements are changing states locally

Design Bugs & Limitations

Concurrent Top-down/Bottom-Up

- Bugzilla #760532, #768522, #759604
- 2nd async state change is ignored by GstBin
  - It's not clear what to do! Who/what has priority?
- Can lead to (at least)
  - Elements stuck in PAUSED forever
  - Elements stuck in PLAYING although bin is PAUSED
  - Base time is not set correctly due to no `change_state()`
- Usually not a problem: both application triggered and both entry points take the state lock!

Mixing live and async

- Bugzilla #760532
- READY → PAUSED
  - NO_PREROLL overrides ASYNC, ASYNC is forgotten
  - Commit state immediately
- Later state change forgets NO_PREROLL
- Losing state can cause whole pipeline to stay ASYNC in PAUSED
- Causes inconsistencies or pipeline stuck in PAUSED

- Inconsistencies usually ignored, stuck in PAUSED very unlikely

Inconsistencies & Debuggability

- State lock is used inconsistently, different than the docs say
  - There are probably some hidden bugs here
  - Fixing it is not possible at this point as it breaks existing code

- Having state changes handled from multiple threads concurrently makes debugging borderline impossible
  - It's also often not clear what the correct behaviour should be if there are multiple concurrent ones

Ideas for a better future

Always let the pipeline handle it

- Top-down, always
- Dedicated, single thread, properly locked
  - No concurrent state changes
  - One after another
- Lost state by asking the pipeline
  - No magic state value changing
  - Always go through normal state change machinery

Always Asynchronous

- Dedicated thread makes it easy to do everything asynchronous
- Simpler code in general
  - Only need to handle ASYNC case
- Nobody wants synchronous (blocking!) state changes anyway

Don’t mix live’ness with states

- Track separately from the state
- Always commit states immediately if any live element in the bin
- Do asynchronous locally in the other elements

- Like now but more consistent and not forgetting live'ness

Thanks!

Any questions?