
**Tan Ngoc Do**

Renesas Design Vietnam Ltd., Co.
Software Department

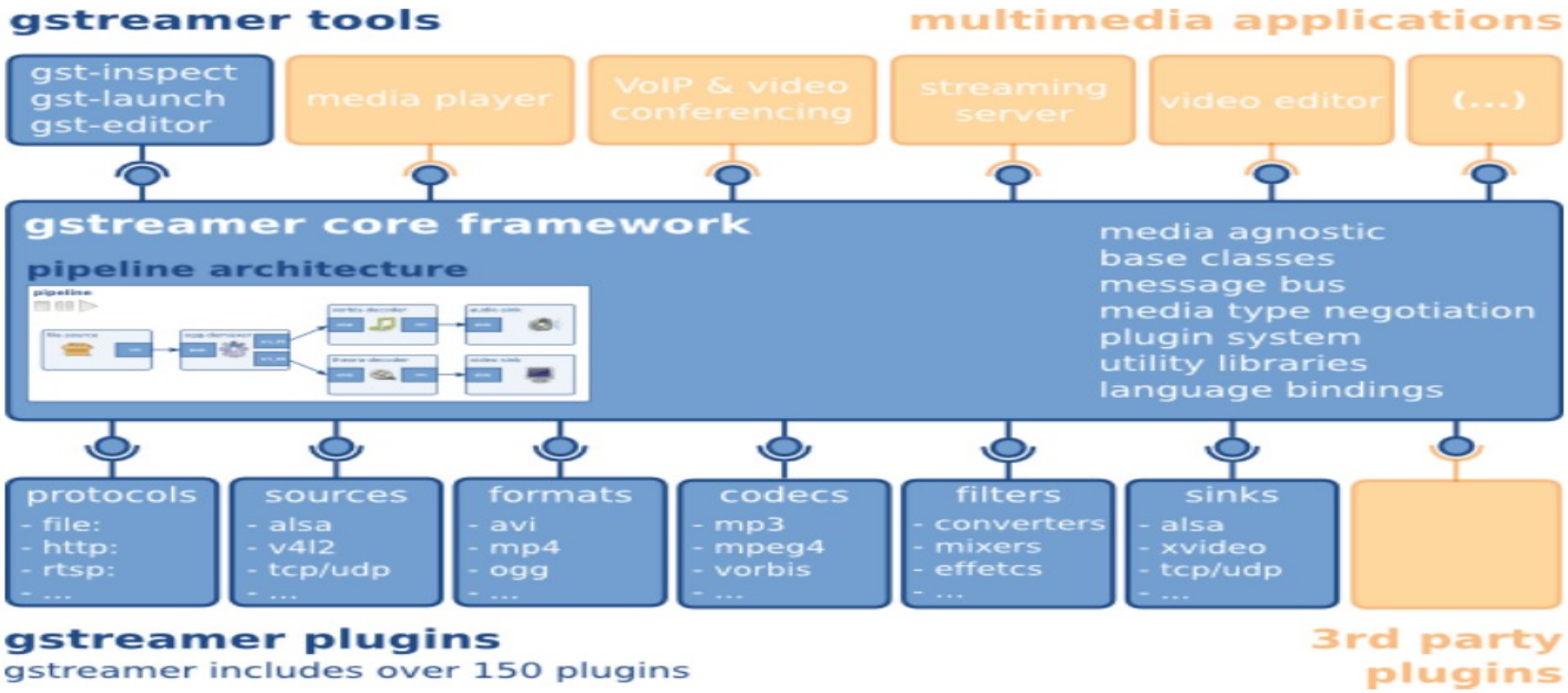
Jul 11th, 2012

# Contents

- Introduction
- Background Knowledge
- Communication
- GStreamer Plug In

# Introduction

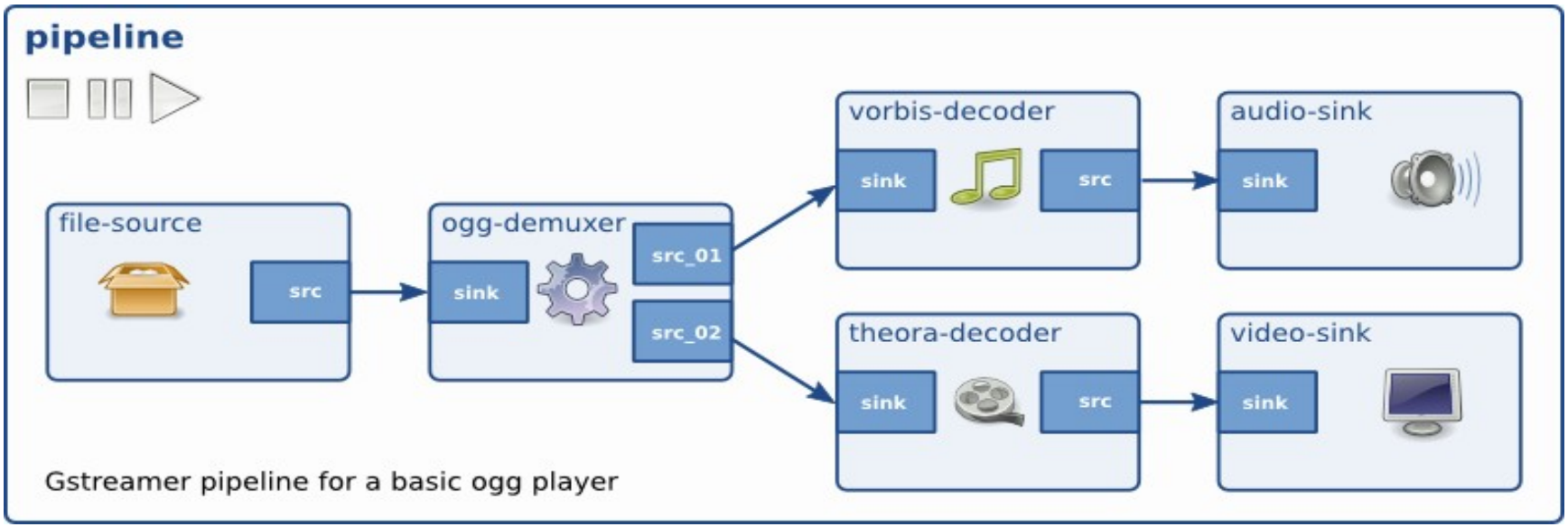
- **GStreamer :**
  - A framework for creating streaming media applications
  - Built base on some ideas of DirectShow
  - Type System based on Gobject



# Background

# Elements

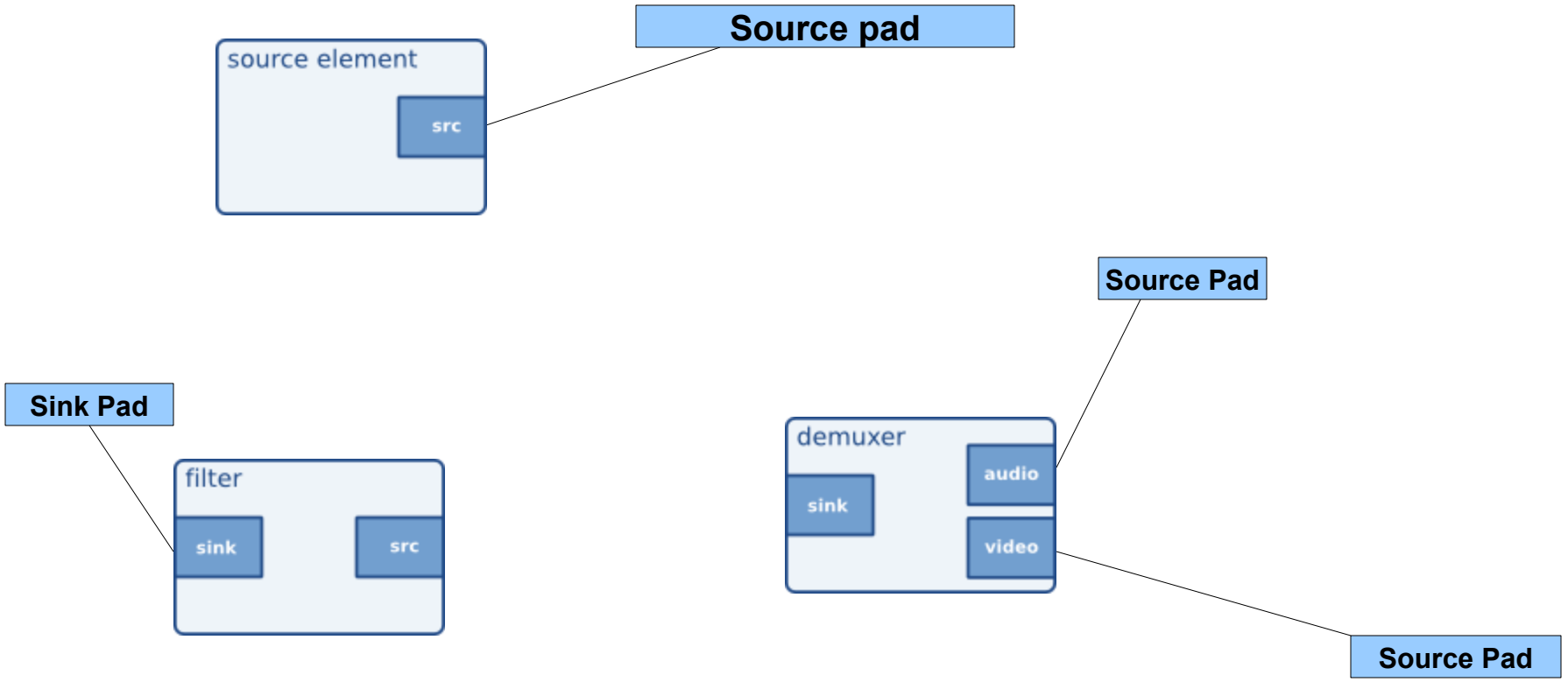
- Most important class of objects in Gstreamer
- Use **GstElement** to describe
- Types:
  - Source elements
  - Filters, convertors, demuxers, muxers and codecs
  - Sink elements
- Chaining together several  elements to create a ***pipeline***



# Pad

- Element's input and output
- Where Elements transfer data and connect to others
- Describe by GstPad
- Categorize:
  - According to Directions
  - According to availabilities

# Directions of pad



# Availability of Pad

- Dynamic(sometimes)
  - Elements might not have all of their pads when created
  - Ex:demuxer creates audio and video pad when demuxing and delete after finish
- Request Pad:
  - Pad is created on demands
  - Ex:multiplex create number of source pad (for output data ) bases on requested number source pad
- Always

# Capabilities of Pad

- Describe by GstCap.
- **Describe the types of media that may stream over a pad**
- Contain some structure(GstStructure) to specify data types

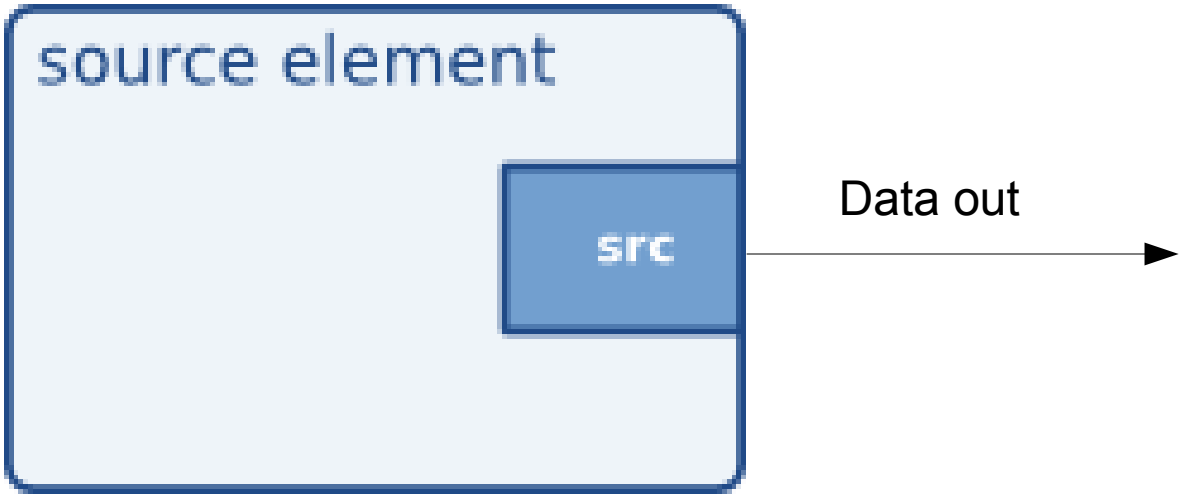
```
Pad Templates:
  SRC template: 'src'
    Availability: Always
    Capabilities:
      audio/x-raw-float
                    rate: [ 8000, 50000 ]
                channels: [ 1, 2 ]
              endianness: 1234
                   width: 32
            buffer-frames: 0

  SINK template: 'sink'
    Availability: Always
    Capabilities:
      audio/x-vorbis
```

# Caps Negotiation

- The process where elements configure themselves and each other for streaming a particular media format over their pads
- After negotiating successfully data can be transfer between 2 pads

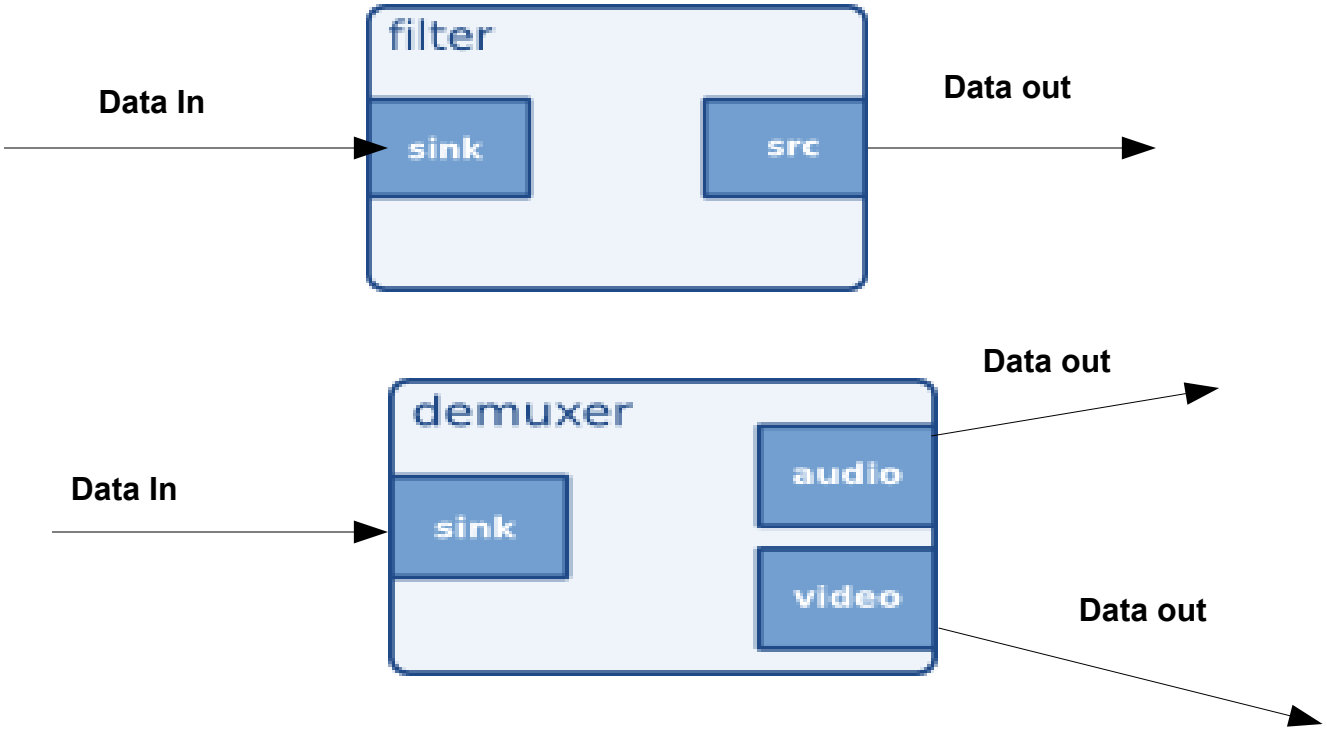
# Source Element

- Generating data for use by a pipeline
  - EX:reading from disk or from a sound card or from live stream
- Do not accept data
- **Only has one source pad**



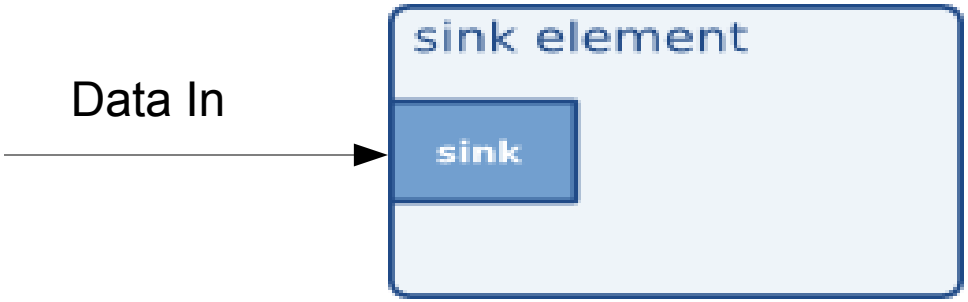
# Filters, convertors, demuxers, muxers and codecs

- Have both sink pad and source pad
- Can have any number of source or sink pads
- Manipulate received data from **sink pad** and produce data to **source pad**
-



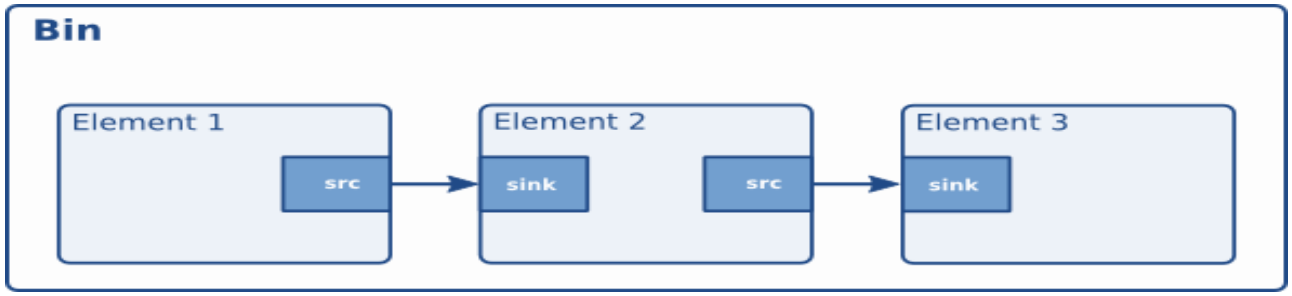
# Sink elements

- End points in a media pipeline
- Accept data but do not produce anything
- EX:Disk writing, soundcard playback, and video output



# Bins and Pipeline

- A bin is a container for a collection of elements(GstBin)




- A pipeline(GstPipeline) is a special subtype of a bin that allows execution of all of its contained child elements
- Bin is a subclass of element
- The relationship:

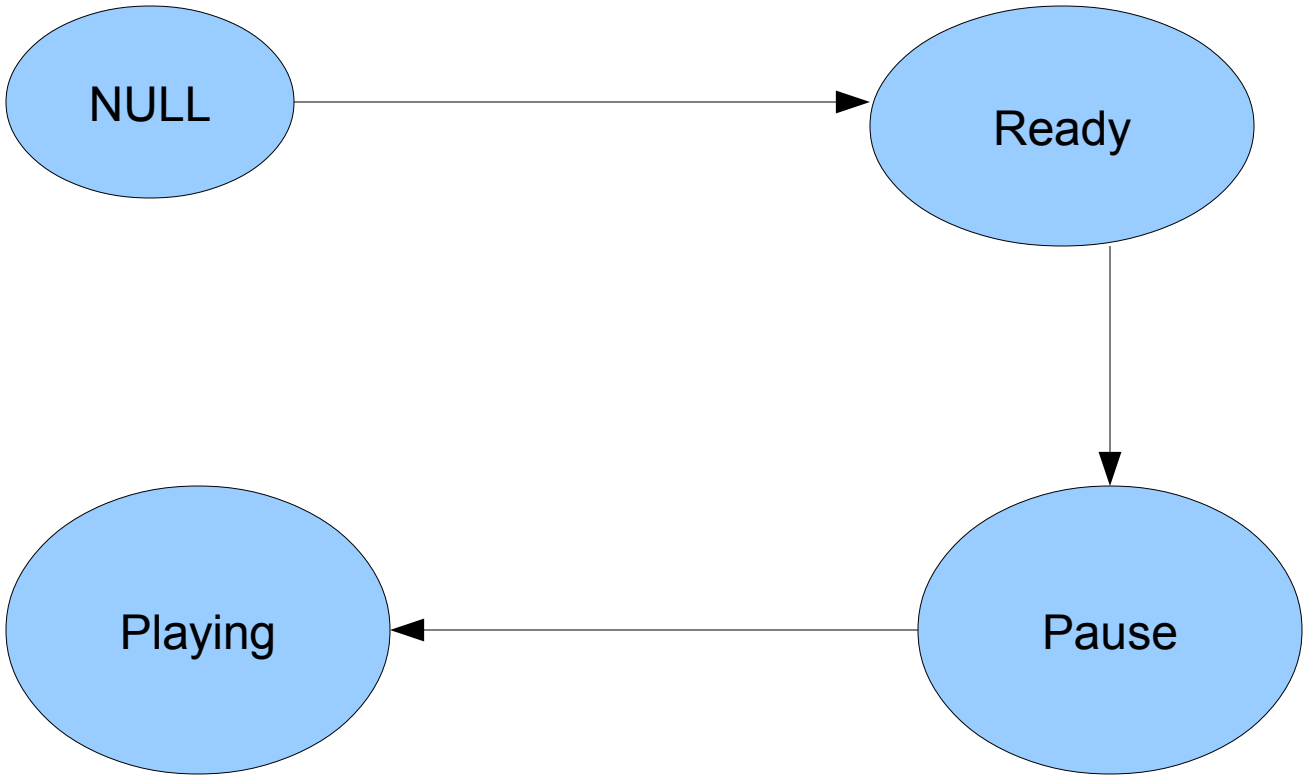
```
+----GstElement
      +----GstBin
            +----GstPipeline
```

# States of Element 1-3

- GST_STATE_NULL:
  - No resources are allocated
  - Transitioning to it will free all resources
- GST_STATE_READY:
  - Element has allocated all of its global resources
  - The stream is not opened
  - Stream positions is automatically zero
- GST_STATE_PAUSED:
  - Element has opened the stream but is not actively processing
  - Sink elements only accept one buffer and then block(prerolled)
- GST_STATE_PLAYING:
  - Element has opened the stream
  - Stream is actively processing

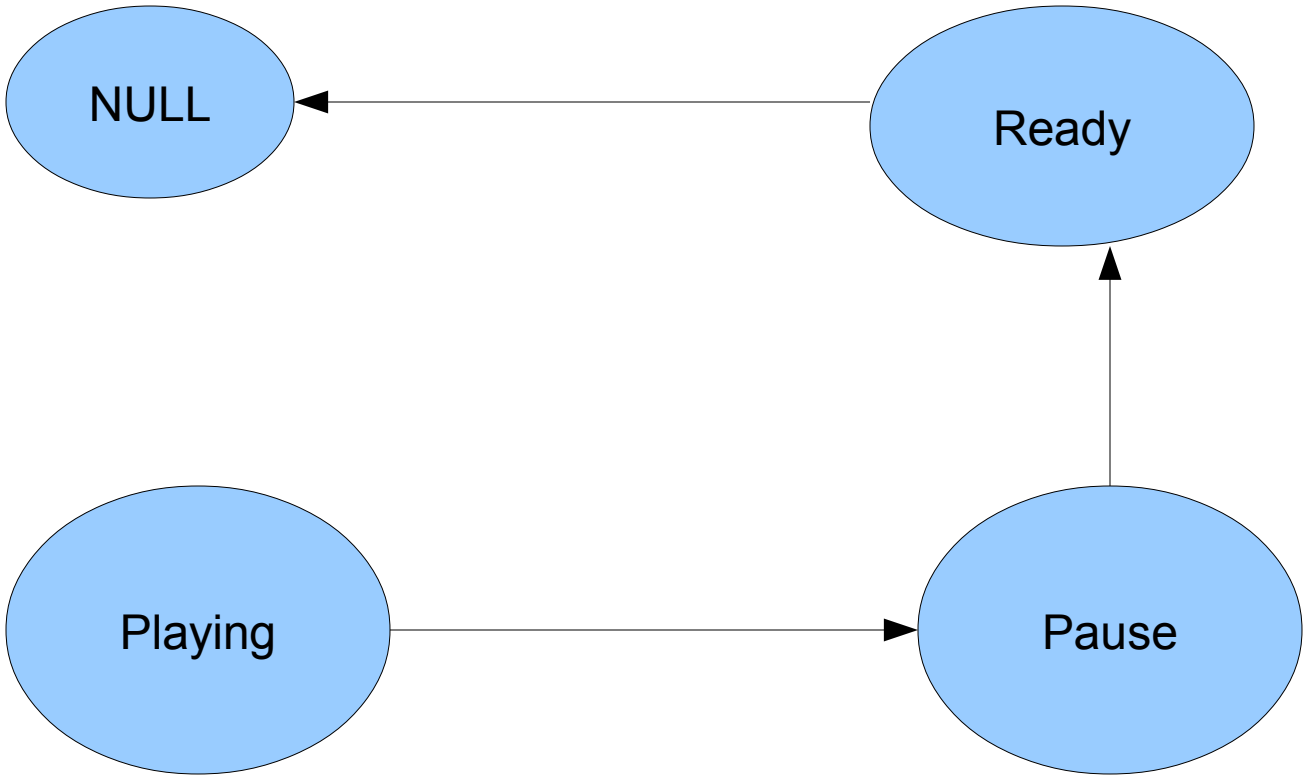
# States of Element 2-3

- Upwards



# States of Element 3-3

- Downwards

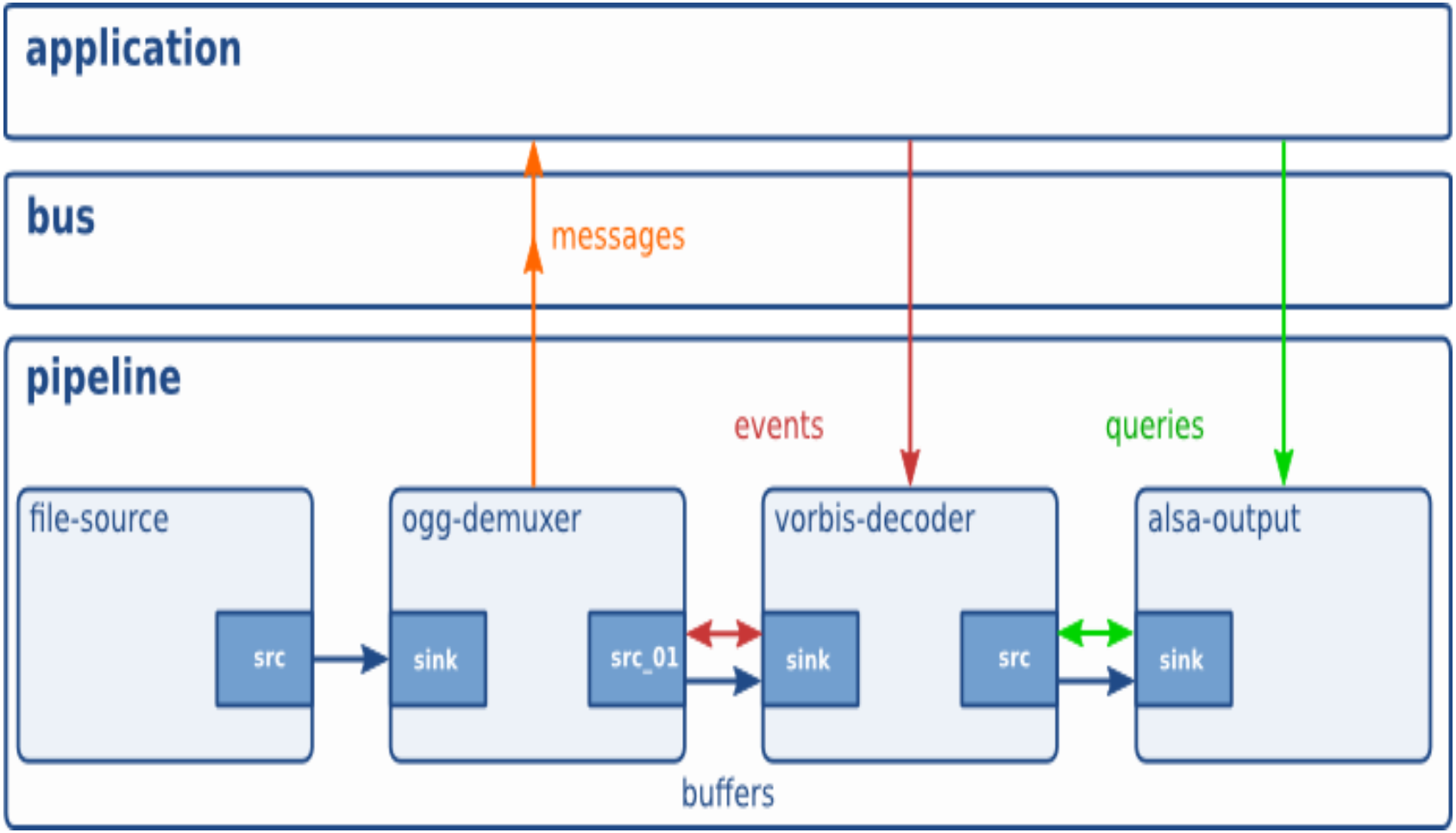


# The pad activation stage

- When changing states from **Ready-Pause**: the pads of an element will be activated to prepare data flow
- This happens from src->sink pad
- Plugins developers take care of writing and register for activated function for each pad(refer Plug-in guide for more detail)

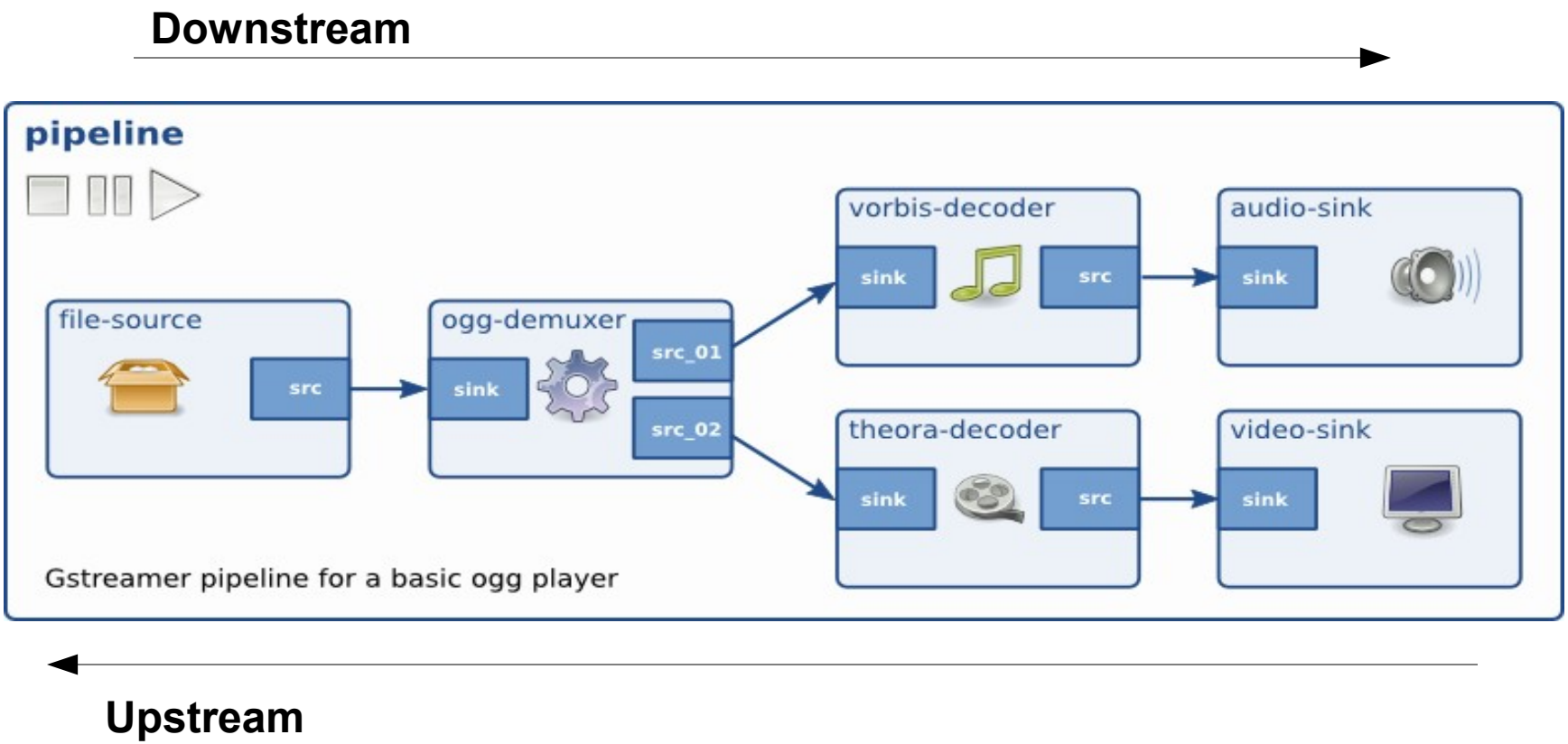
# Communication

# Overview


application
bus
messages
pipeline
events
queries
file-source
src
ogg-demuxer
sink
src_01
vorbis-decoder
sink
src
alsa-output
sink
buffers

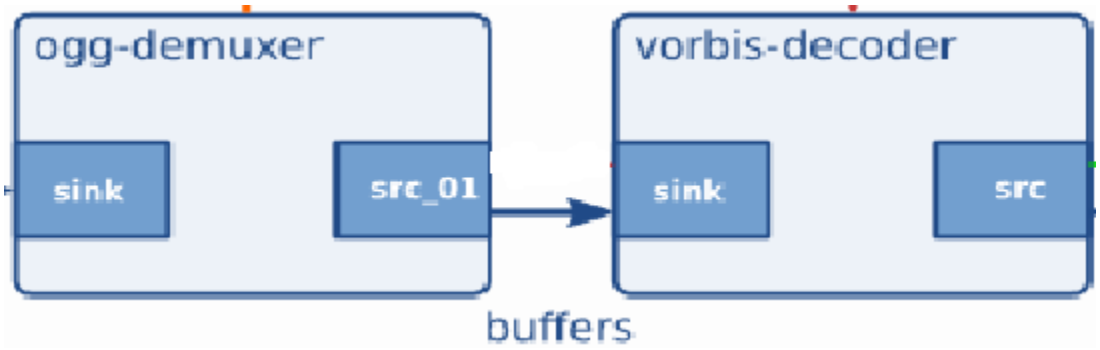
# Downstream and Upstream

- Downstream and upstream are the terms used to describe the direction in the Pipeline
  - Downstream:From source to sink
  - Upstream:from sink to source



# Buffers 1-2

- Objects for passing streaming data between elements in the pipeline
- Always travel from sources to sinks (**downstream**)
- A source element will typically create a new buffer and pass it through a pad to the next element in the chain



# Buffers 2-2

- Application developer does not need to care so much about the buffer
- Plug-in developer needs to implement buffer
- Buffer can consist :
  - A pointer to a piece of memory
  - The size of the memory
  - A timestamp for the buffer
  - A refcount that indicates how many elements are using this buffer
  - Buffer flags

# Events

- Events are objects sent:
  - Elements ->elements(**down stream or up stream**)
  - Application->Elements
- Plug-In developers need to care both downstream and upstream events

# BUS

- Simple system that takes care of forwarding messages from the pipeline threads to an application in its own thread context
- Describe by GstBus
- Application just “listen messages” from the pipeline.

# Message

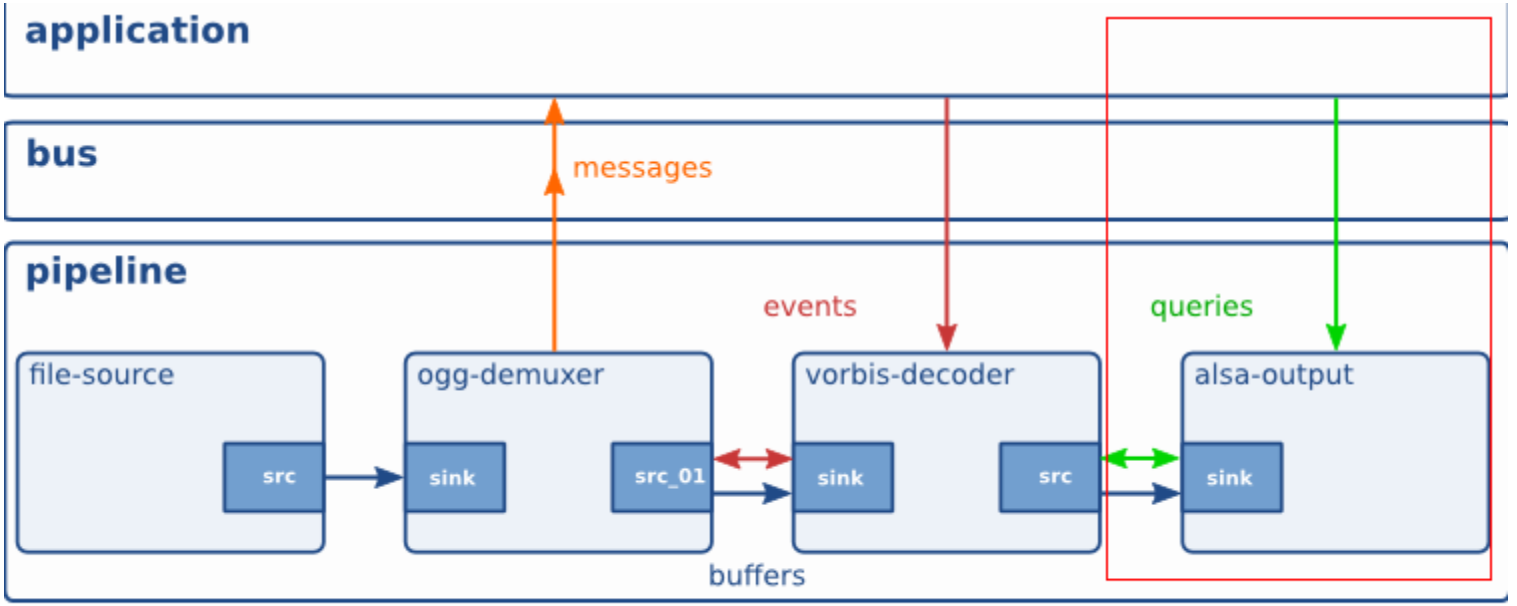
- Objects posted by elements on the pipeline's message bus to application
- Consist:
  - Errors
  - States of pipeline
  - End of stream.....

# Messages

- Some Messages:
  - GST_MESSAGE_UNKNOWN:undefined message
  - GST_MESSAGE_EOS:end of stream in pipeline
  - GST_MESSAGE_ERROR: an error occurs
  - GST_MESSAGE_WARNING:an warning occurs

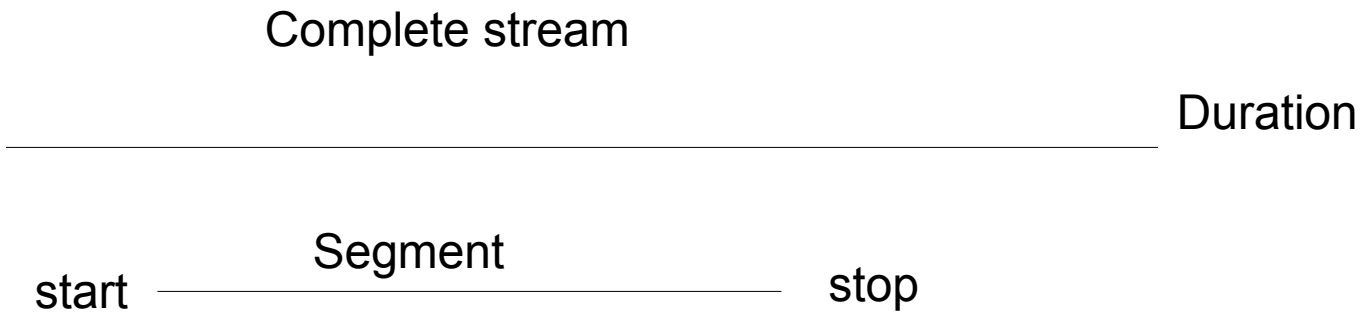
# Queries

- Allow applications to request information such as duration or current playback position from the pipeline



# Segment

- Denotes a set of media samples that must be processed
- Consist of:
  - Start time
  - Stop time
  - Processing rate
- Application can seek to segments and play them by calling **gst_element_seek**



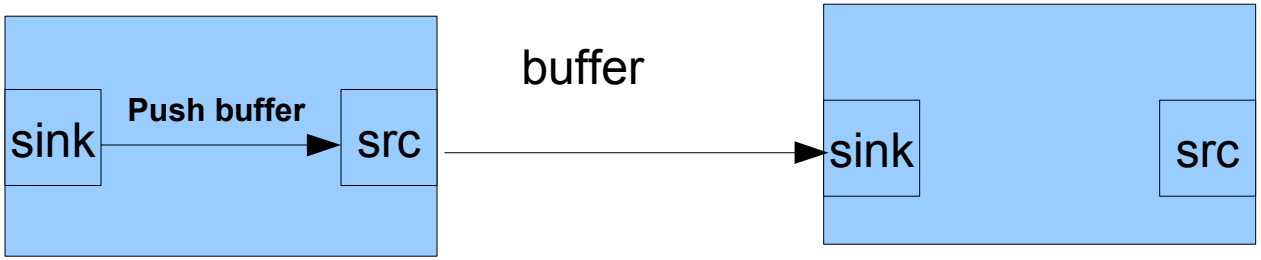
# Scheduling modes 1-3

- The scheduling mode of a pad defines how data is retrieved from (source) or given to (sink) pads.

- Consist of:
  - Pushing-mode
  - Pulling-mode

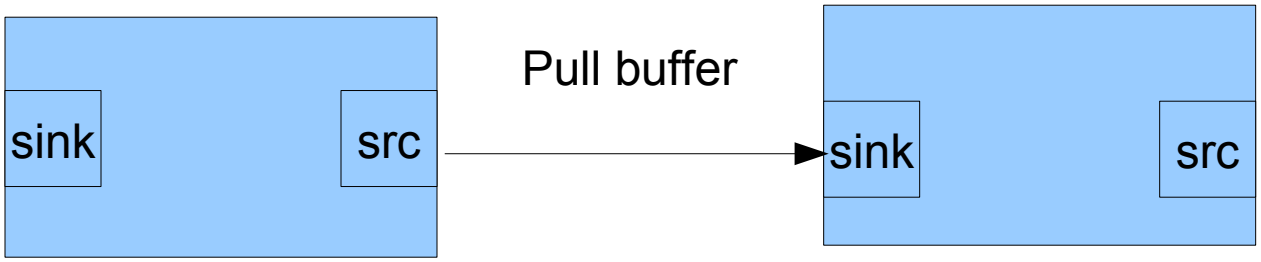
# Scheduling modes 2-3

- Pushing-mode:A pad can produce data and push it to the next pad
- Upstream element **pushes** data to downstream element
- In this mode src pad is a driving force for data flow on pipeline

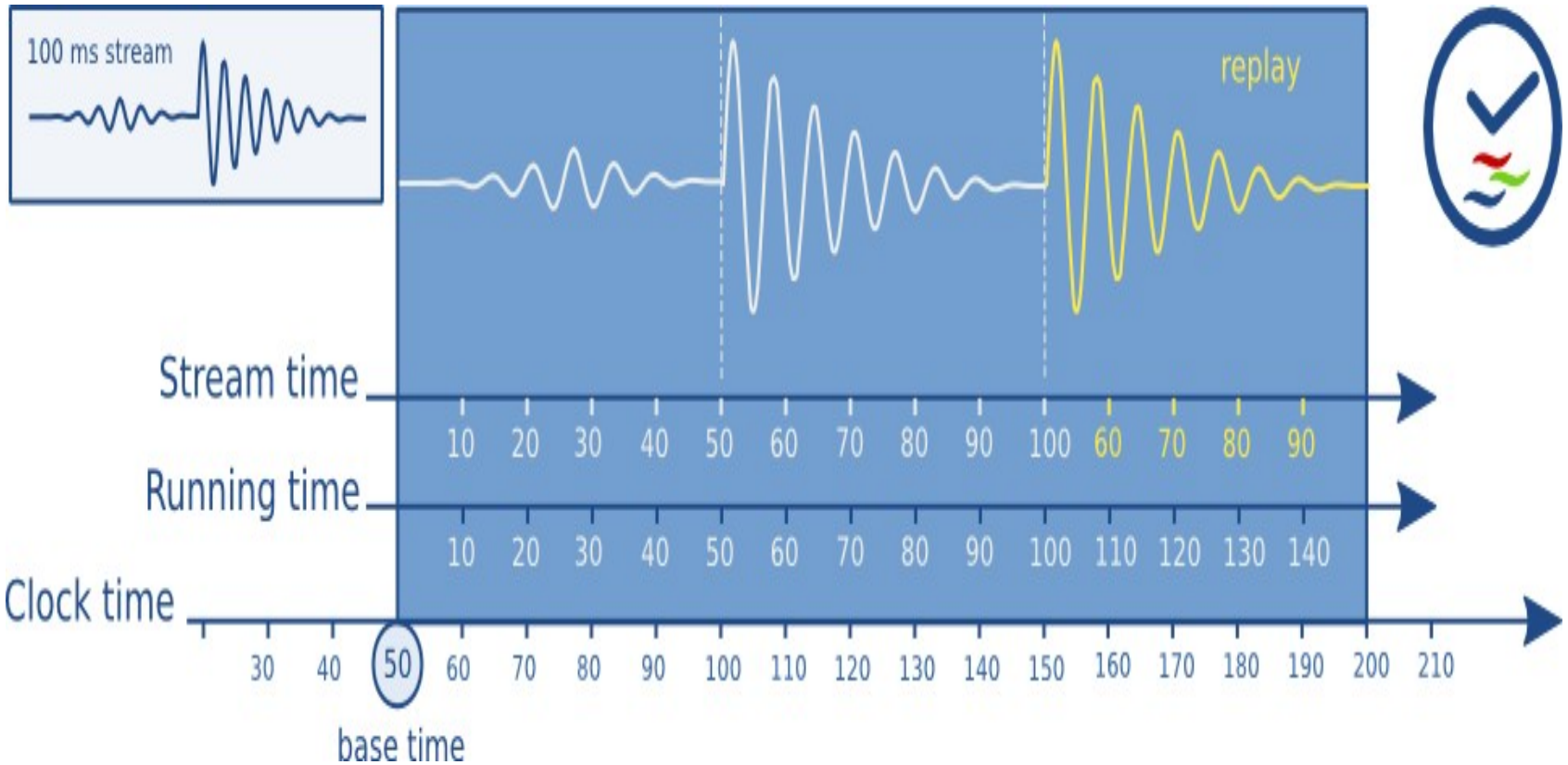


# Scheduling modes 3-3

- In this mode downstream element can **pull** data from an upstream element



# Time Overview



# Time Overview-Clock Time 1-2

- Clock Time(absolute time):GstClock provides source counter to represent the current time in nanoseconds
- Different sources:
  - The system time (with g_get_current_time() and with microsecond accuracy)
  - An audio device (based on number of samples played)
  - A network source
- GstPipeline object maintains a GstClock object and a base-time when it goes to the **PLAYING** state

# Time Overview-Clock Time 2-2

- Before the pipeline is set to PLAYING, the pipeline asks each element if they can provide a clock
- The clock is selected in the following order:
  - If the application selected a clock, use that one
  - If a source element provides a clock, use that clock
  - Select a clock from any other element that provides a clock, start with the sinks
  - If no element provides a clock a default system clock is used for the pipeline

# Time Overview-Clock Running Time

- After a pipeline selected a clock it will maintain the running_time based on the selected clock. This running time represents the **total time spent in the PLAYING state**
  - If the pipeline is NULL/READY, the running_time is undefined
  - In PAUSED, the running_time remains at the time when it was last PAUSED
  - When the stream is PAUSED for the first time, the running time is 0

# Time Overview-Stream Time

- Also known as the position in the stream
- Represents the time inside the media as a value between 0 and the total duration of the media
- Used in:
  - Report the current position in the stream with the POSITION query
  - The position used in the seek events and queries

# GStreamer Plugin

- Good plug-in:
  - High quality plug-ins under the LGPL license
- Bad Plug-in:closely approach good-quality plug-ins but lack
  - A good code review
  - Some documentation
  - A set of tests
  - A real live maintainer
  - Some actual wide use
- Ugly Plug-in:
  - Set of good-quality plug-ins that might pose distribution problems

# The End