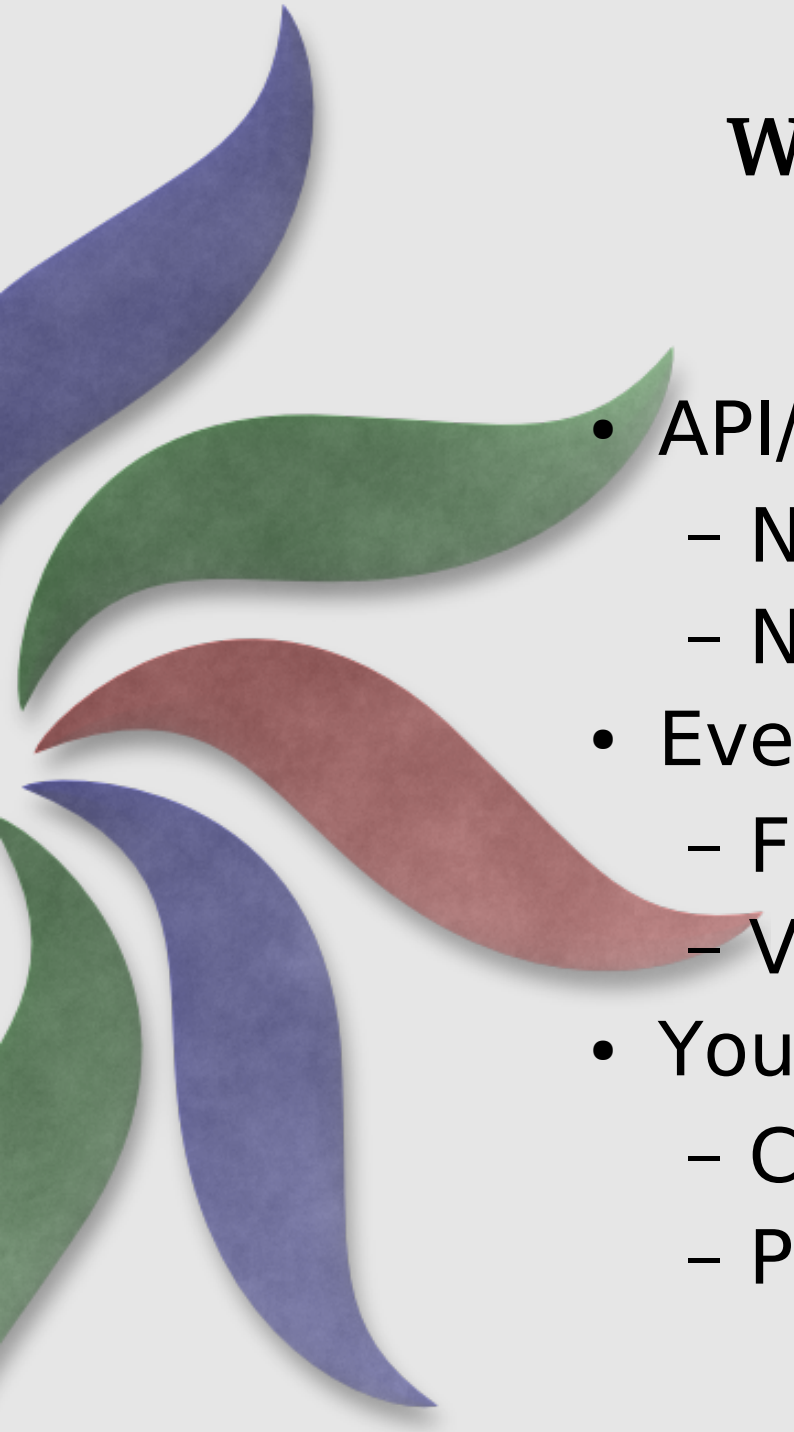
GStreamer
More than just playback

Michael Smith
msmith@xiph.org

linux.conf.au 2008, Melbourne

Introduction to GStreamer

- What is it?
  - A framework for multimedia
  - Similar to DirectShow, QuickTime, etc.
  - Linux, Windows, OS X, ...



What is a multimedia framework

- API/Libraries to write applications
  - Not a codec implementation
  - Not an actual application itself
- Everything provided by plugins
  - Fully extensible
  - Very flexible
- You don't need to know
  - Codec details
  - Platform-specific details

Why not something else?

- Good quality documentation
- Well-designed API (many years of experimentation!)
- API and ABI stability
- Flexible architecture
- Extensive testsuite
- Bindings to multiple languages:
  - C
  - Python
  - Others: Java, Ruby, Perl, etc.

Why GStreamer?

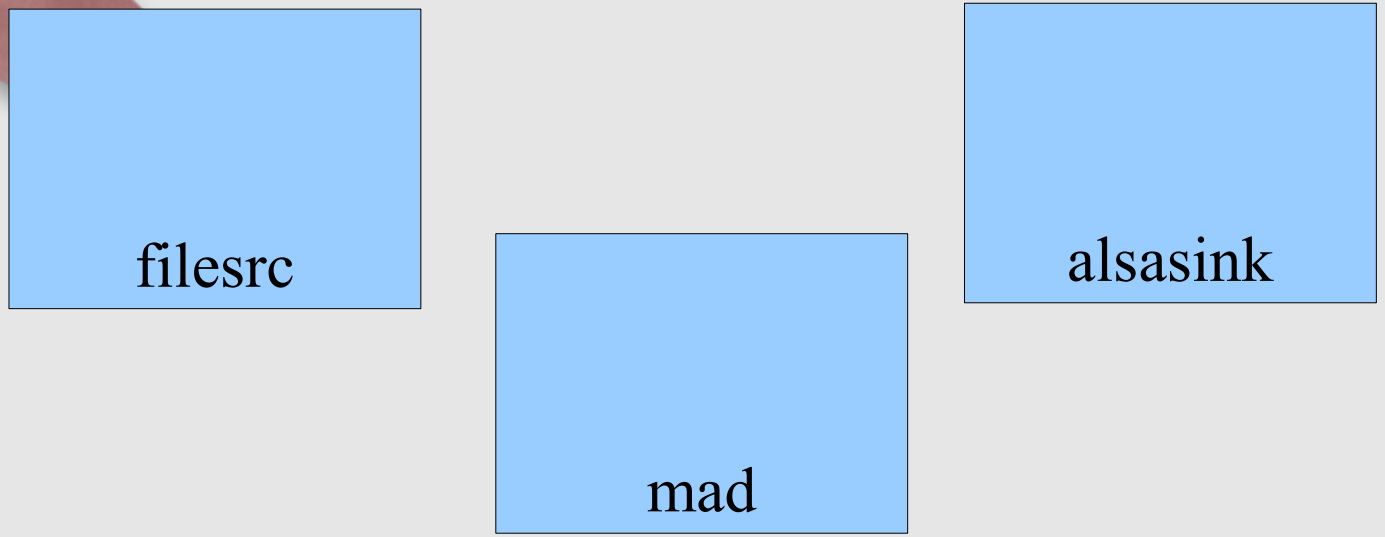
- Flexible platform for all multimedia
- Cross-platform
- Strong community, many contributors
- Commercial support, codecs, etc
- License: LGPL
- Clear legal situation

GStreamer: The Basics

- Media flow graph: Pipeline
- Elements, Bins, Pads, Caps

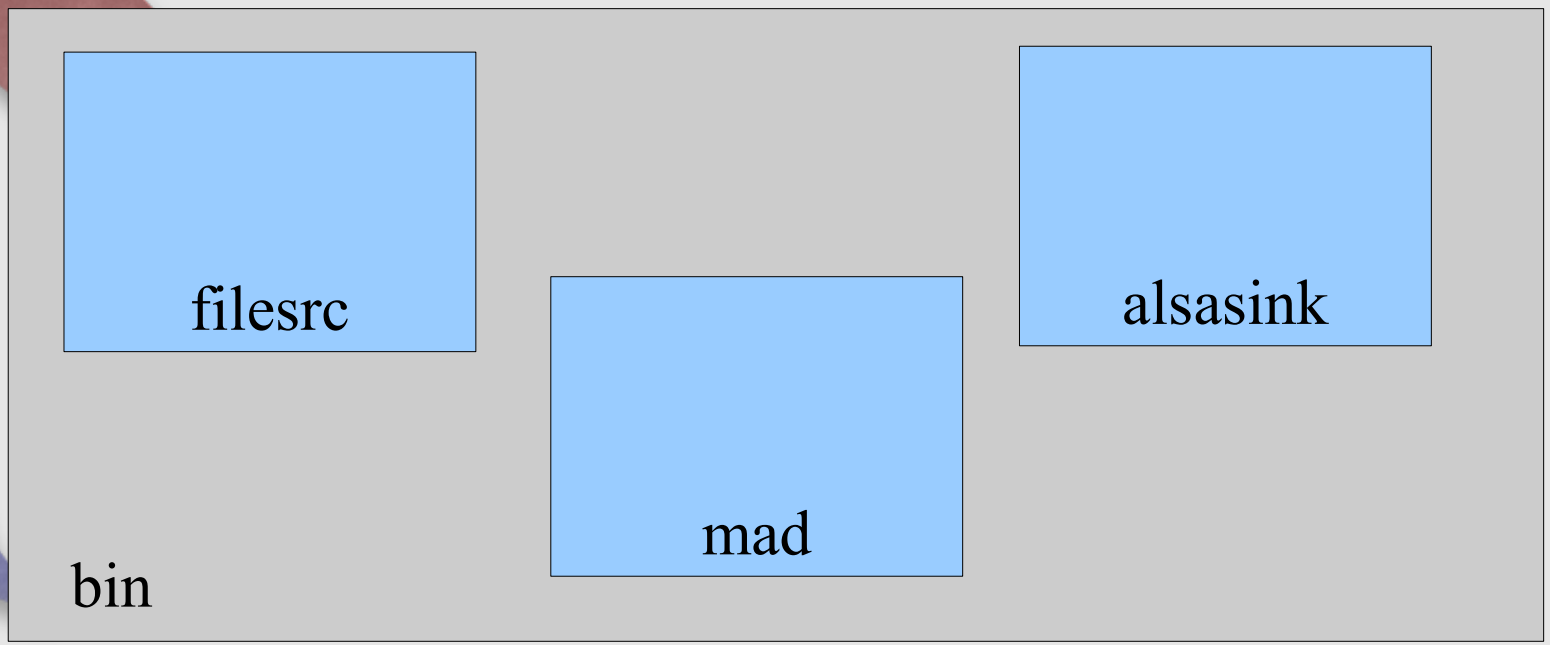
Basic parts of GStreamer

- Elements
  - The basic units of functionality



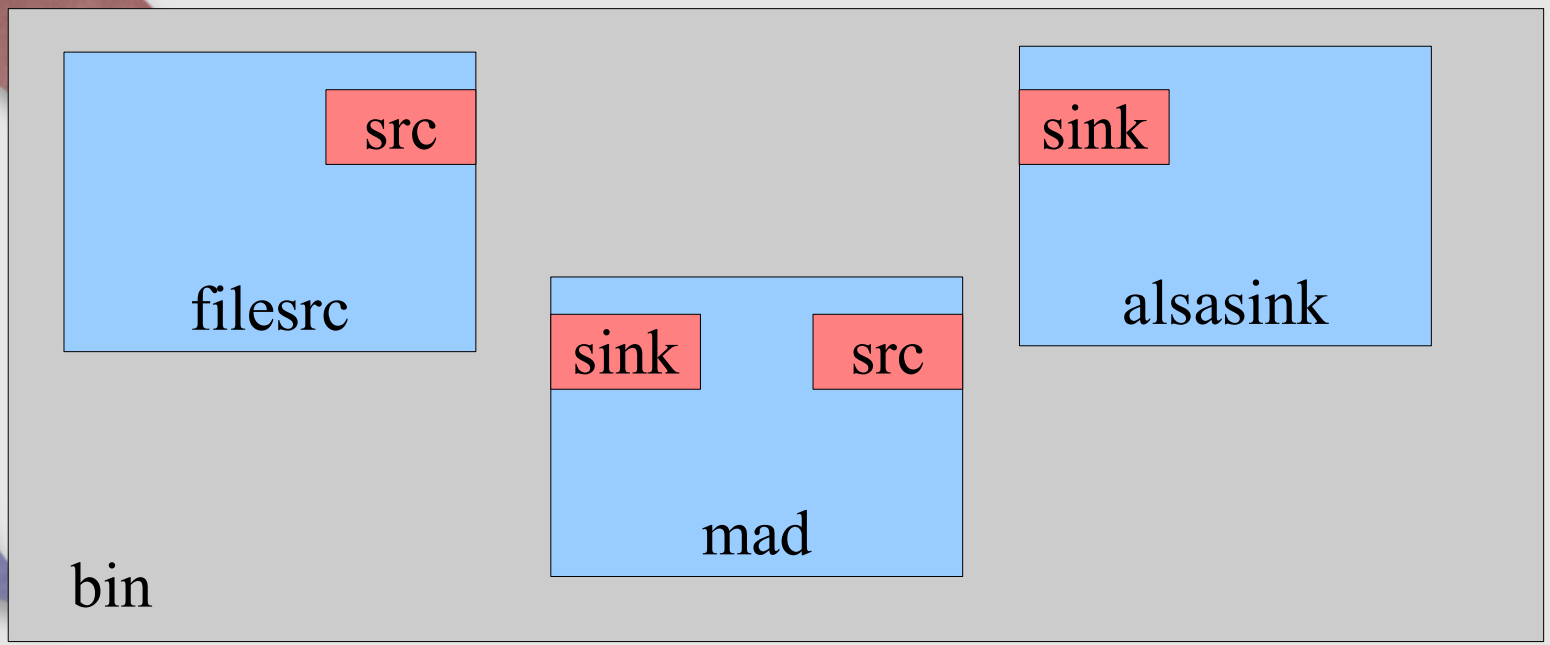
Basic parts of GStreamer

- Bins
  - Elements that contain other elements
  - Allow multiple elements to be treated as one entity
  - Pipeline is a top-level GstBin



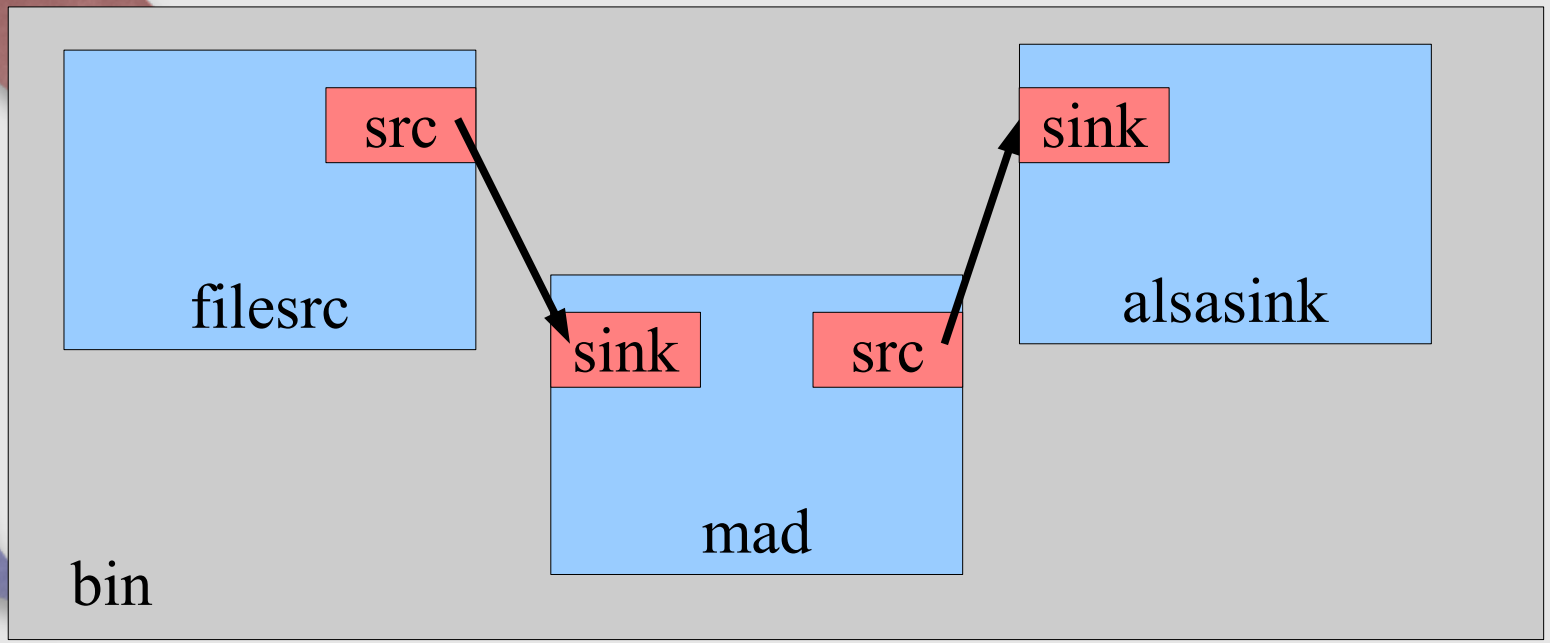
Basic parts of GStreamer

- Pads
  - Connection points between elements
  - Dataflow only happens between src/sink pad pairs



Basic parts of GStreamer

- **Source** pads produce data
- **Sink** pads consume data



Data types and Caps

- Data types in pipeline described by 'Caps' (GstCaps object)
- Type name with name/value properties
- Text representation:
  - `audio/x-raw-int, rate=44100, depth=16, channels=2`
- `gst-inspect` shows these
  - Examples: `vorbisenc`, `xvimagesink`

Negotiation and autoplugging

- Elements can negotiate with each other on format to use
  - Example: video decoder, converter, video sink
- Autoplugging:
  - Typefinders
  - Registry lists all available elements, and possible caps for each
  - Automatically select appropriate demuxers, decoders, etc.

Dataflow: Buffers, Events

- Buffers contain data, have caps
- Flow downstream
- Events flow both down and upstream
  - EOS, seek, tags, QoS, ...
- Both flow across pads, between elements

Message Bus

- GstBus: message delivery to your application
- Main communication channel from pipeline to application
- Avoiding worrying about threads!

So what can we do with this?

- Individual plugins:
  - Encoding, decoding, muxing, demuxing
  - Sources, sinks
- Higher-level functionality (bins):
  - decodebin
  - playbin
  - rtspsrc
  - gnonlin
- gst-launch as a developer tool

Writing applications

- Players
  - Rhythmbox, Totem, ...
- Encoders
  - Thoggen, Sound Juicer
- Servers
  - Flumotion
- Editors
  - PiTiVi, Jokosher
- Anything with multimedia!

Advanced GStreamer

- So, on to the main part of the talk!
- Lots of other libraries/applications give you parts of what GStreamer does
  - Playback (vlc, mplayer, xine)
  - Encoding (ffmpeg, mencoder)
  - Streaming (icecast, etc)
- GStreamer lets you do all of these, plus plug functionality together to make new systems
- Examples, demos, ideas

Clocking and Sychronisation

- Some tricks with playback first
- GstClock
  - Abstract clock API for sychronisation and timing
  - Sinks use this for sync
  - Pipeline normally selects an appropriate clock (audio, system)
  - You can provide a different clock

Network clocks

- GstNetTimeProvider: exposes any GstClock over a network (UDP)
- GstNetClientClock: creates a clock based on a GstNetTimeProvider
- Synchronise any number of clients to the same time provider

Easy to use

- Simple API:

```
GstPipeline *pipeline = (GstPipeline *)
  gst_parse_launch("playbin uri=file:///tmp/test.ogg");
GstClock *clock = gst_net_client_clock_new("netclock",
  "127.0.0.1", 1234, basetime);
gst_pipeline_set_new_stream_time(pipeline,
  GST_CLOCK_TIME_NONE);
gst_pipeline_use_clock(pipeline, clock);
gst_element_set_state(pipeline, GST_STATE_PLAYING);
...
```

- Demo!

More sychronisation tricks!

- `gst_element_seek(pipeline, rate, format, flags, cur_type, cur, stop_type, stop);`
  - 'rate'? What can I do with that?
  - 'flags': accurate, keyframe, flush, etc
  - cur and stop: playback of sections
- Demo!

Encoders and Streaming

- Wide range of encoders (but use free ones!) and muxers
  - Usage example:
    - `gst-launch filesrc location=file:///tmp/test.ogg ! decodebin ! audioconvert ! goom ! ffmpegcolorspace ! theoraenc ! oggmux ! filesink location=/tmp/out.ogg`
- Sources: get data from cameras, TV (DVB, V4L), streams, more...
- Sink plugins for servers (not the whole server)
- Metadata/tagging interfaces
- RTP stack

Editing with GStreamer

- GNonLin: non-linear editing plugins for GStreamer
- gnlcomposition
  - Timeline
  - A bin: contains other elements
  - Controls switching automatically

GNonLin

- gnlsource – a single media source/section
- Can use anything: file, camera, stream, etc.
- Control using properties
  - start
  - duration
  - media-start
  - media-duration
  - priority
- gnlfilesrc

GNonLin Applications

- PiTiVi: Video Editor
  - Not really ready for end-users
  - Demo
- Jokosher: Multitrack audio editor
  - Feedback on ease of development
  - Focus on usability/simplicity