PipeWire

Wim Taymans
Principal Software Engineer
October 2017

In the begining

- Pinos
  - DBus service for sharing camera
  - Upload video and share
  - Using GStreamer for media processing

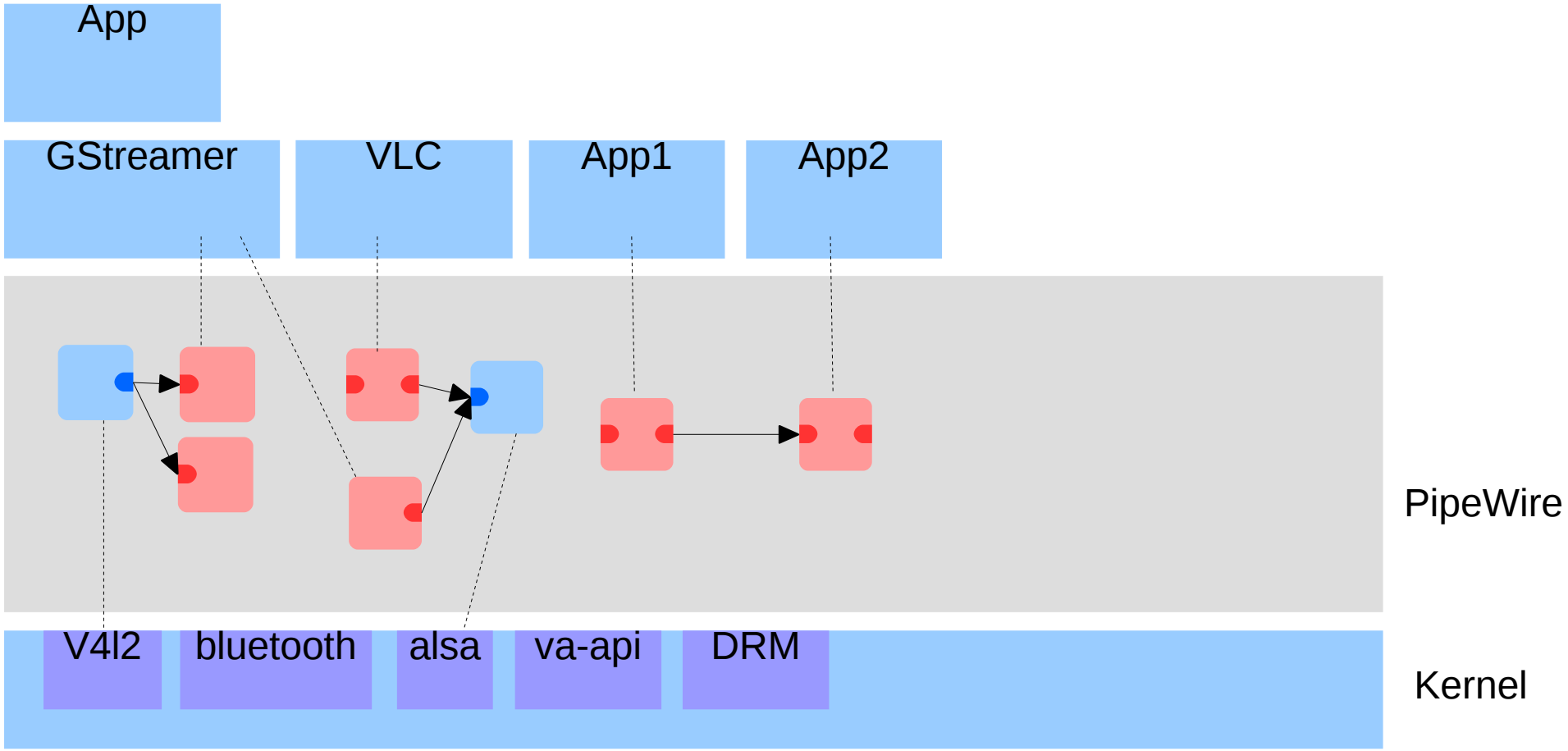
And then...

- Extend scope
  - Add audio too; upload, playback, capture
  - It looked like JACK but without low-latency guarantees
  - Need for real-time processing with extremely low latency 0.3ms <32 bytes buffers
    - Gstreamer + Dbus, can’t we do better
  - Rename to **PipeWire**

What is it

Pipeline based processing engine for multimedia

The multimedia stack



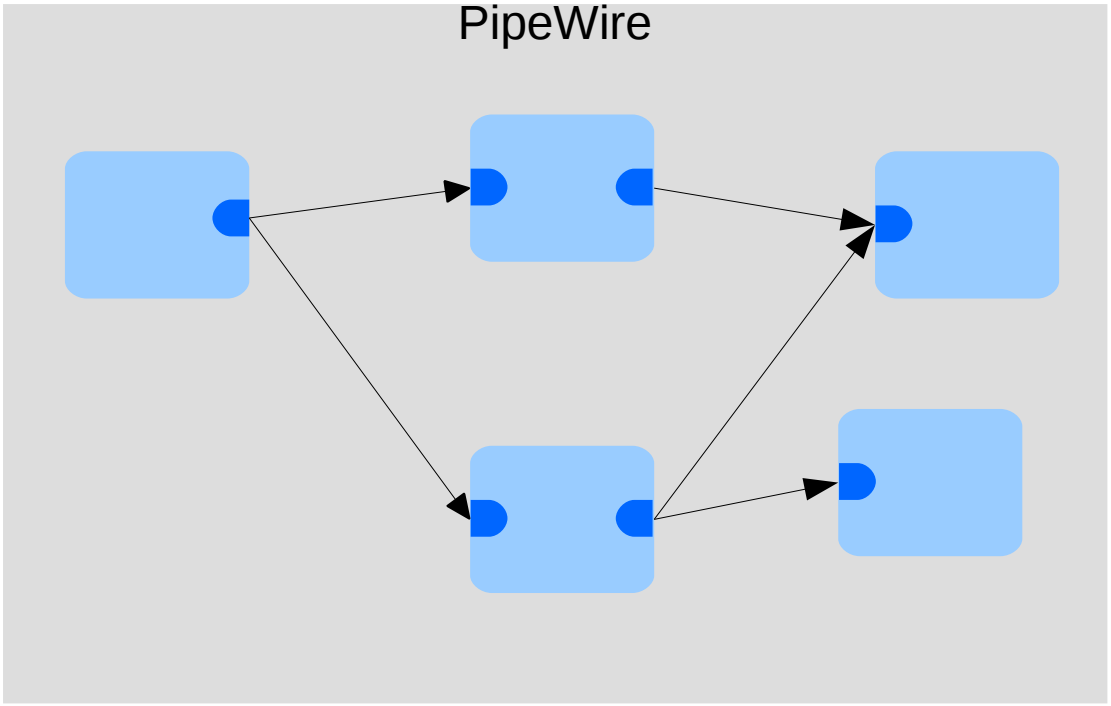
Nodes

- SPA plugins
  - New plugin API
    - Format negotiation
    - Properties/configuration
    - Buffer negotiation
    - Processing
    - Based on LADSPA, LV2, MFT, OpenMax IL, GStreamer
  - Hard RT capable, zero allocations
  - Sync/async operation
  - Very low overhead
  - Extensible

PipeWire library

- Core object maintains list of objects
  - Modules, nodes, factory, clients, links
  - Globals, wrap an object and make it visible to remote clients
- Registry, object that notifies clients about new/removed/changed globals
- Modules extend functionality
- Access control, checks on visibility of core objects and creation of objects
- Groups nodes in a graph, one per process called a PipeWire instance

PipeWire instance

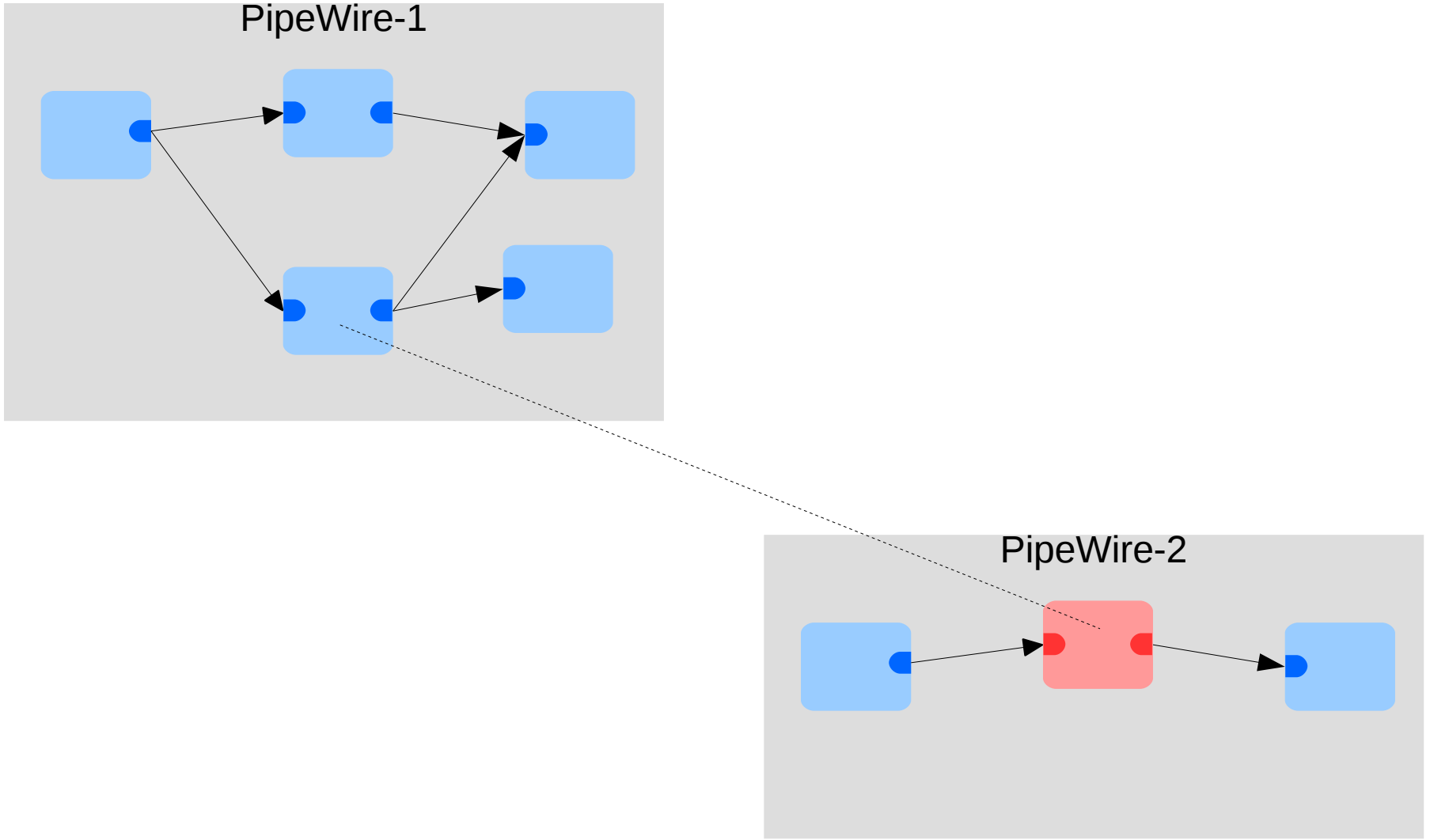


- Connect
- Introspect
- modify

Pipewire instances

Nodes can be shared between instances

Node export



Host for SPA plugins

- Logging, types, allocation
- Loading of plugins
  - Through dynamically loadable modules
- Manages client connections
  - Resources
- Manages global resources
  - Nodes, monitors, factories,...

Resources

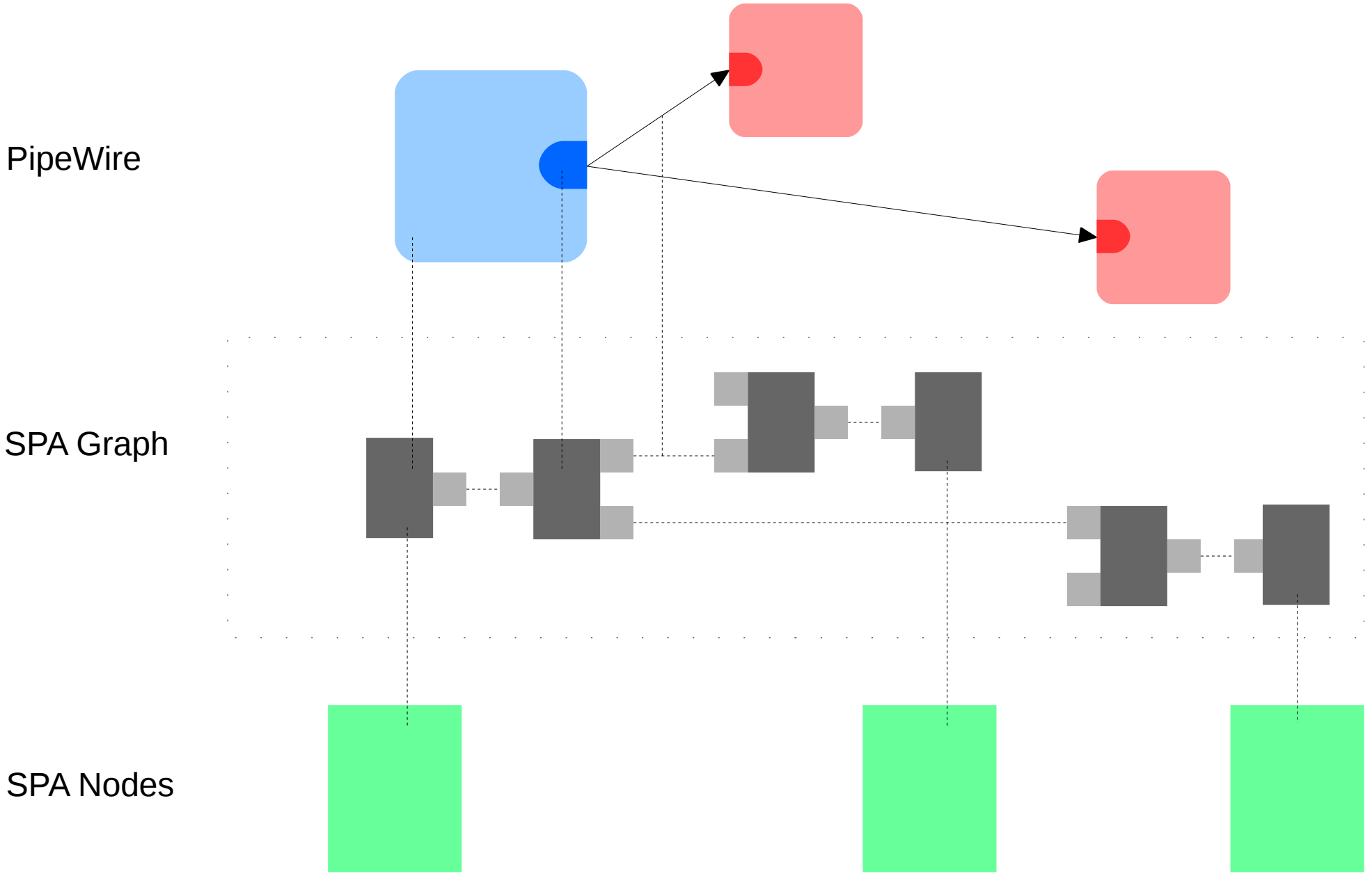
- Clients can communicate with globals by binding to them
- Creates server side Resource object and client side proxy
  - Core object is automatically bound when connecting
  - GetRegistry to get globals
- Exchange messages between the two
  - Methods client → server
  - Events server → client

Nodes, ports

- Nodes are processing objects
  - Wraps SPA node
- Ports are exposed on Nodes
  - Wraps SPA port id
  - Adds mixer and tee SPA Nodes
- Link is connection between ports
  - Negotiates formats
  - Negotiates/allocates buffers
  - Hooks into the port mixer and tee to make a link

Client node

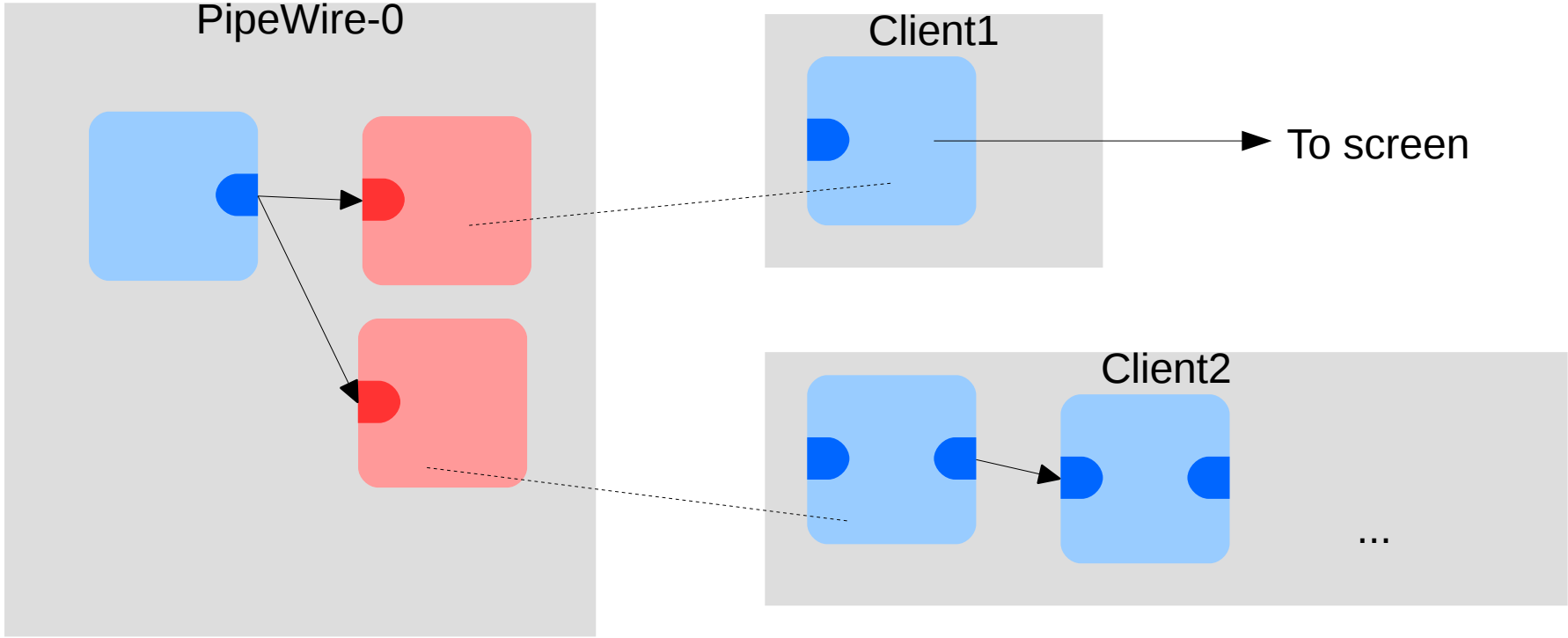
- Client node is Special node
  - Looks like a SPA node
  - All method calls/events are serialized over socket
  - Async replies
- Buffers are allocated in shared memory (memfd)
- Controlled by a remote client


PipeWire
SPA Graph
SPA Nodes

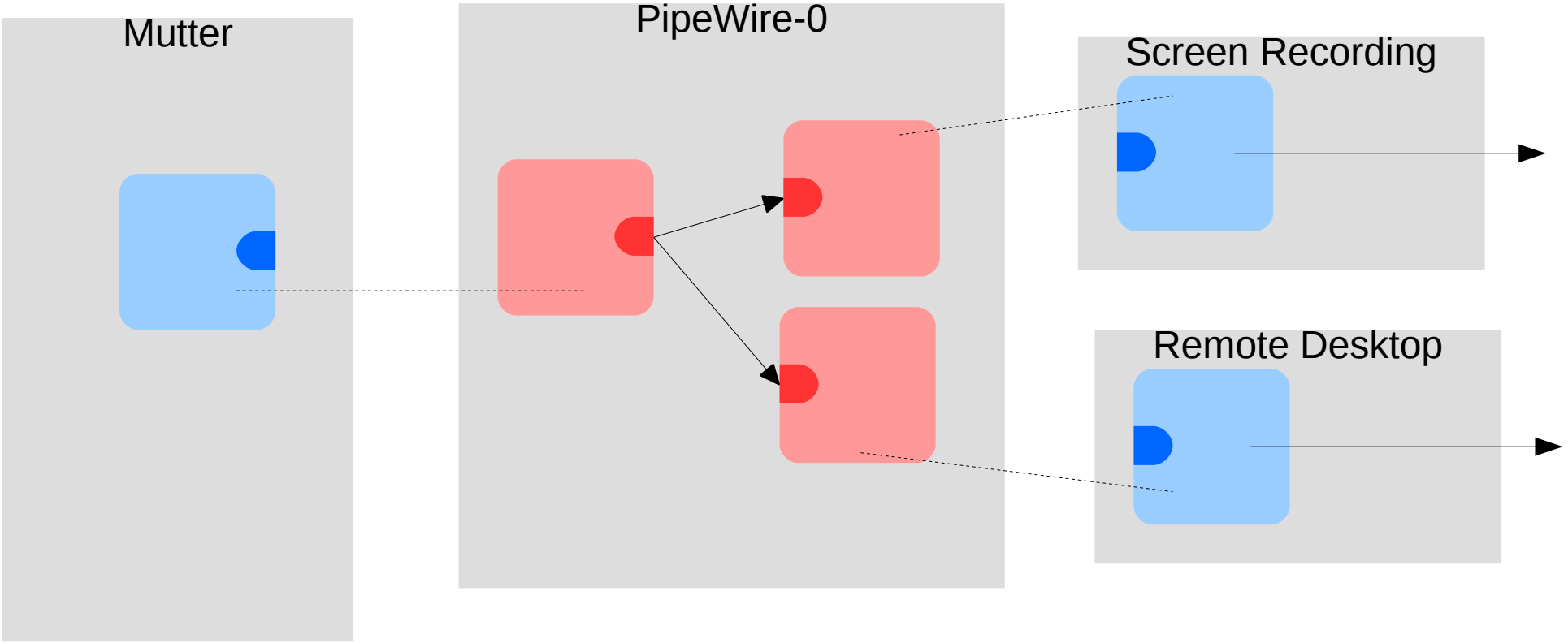
Scheduler

- Operates on the SPA graph
- Runs in RT thread
- Currently events start scheduler
  - Simple push/pull
  - Needs more work

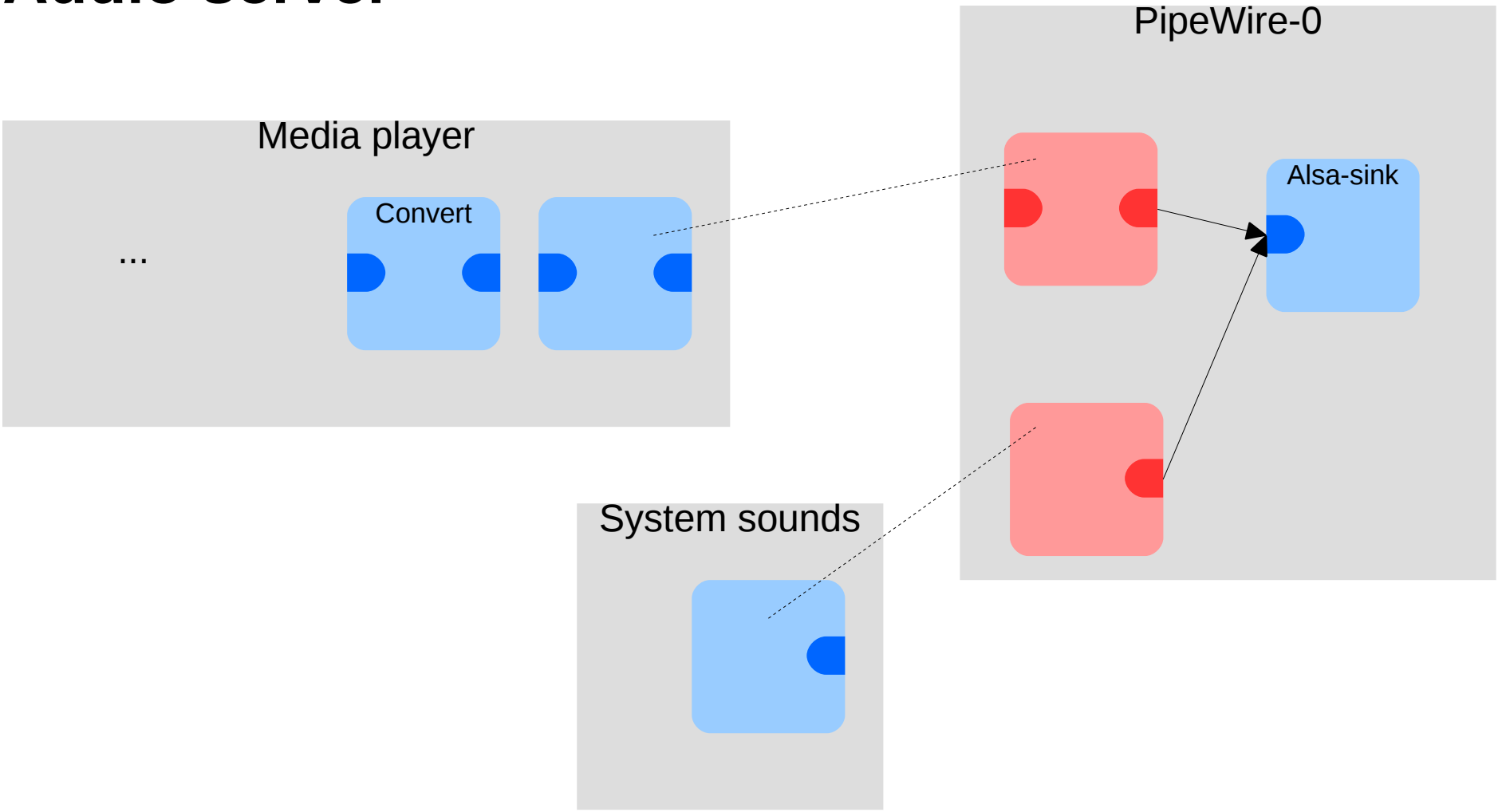
V4l2 capture/sharing



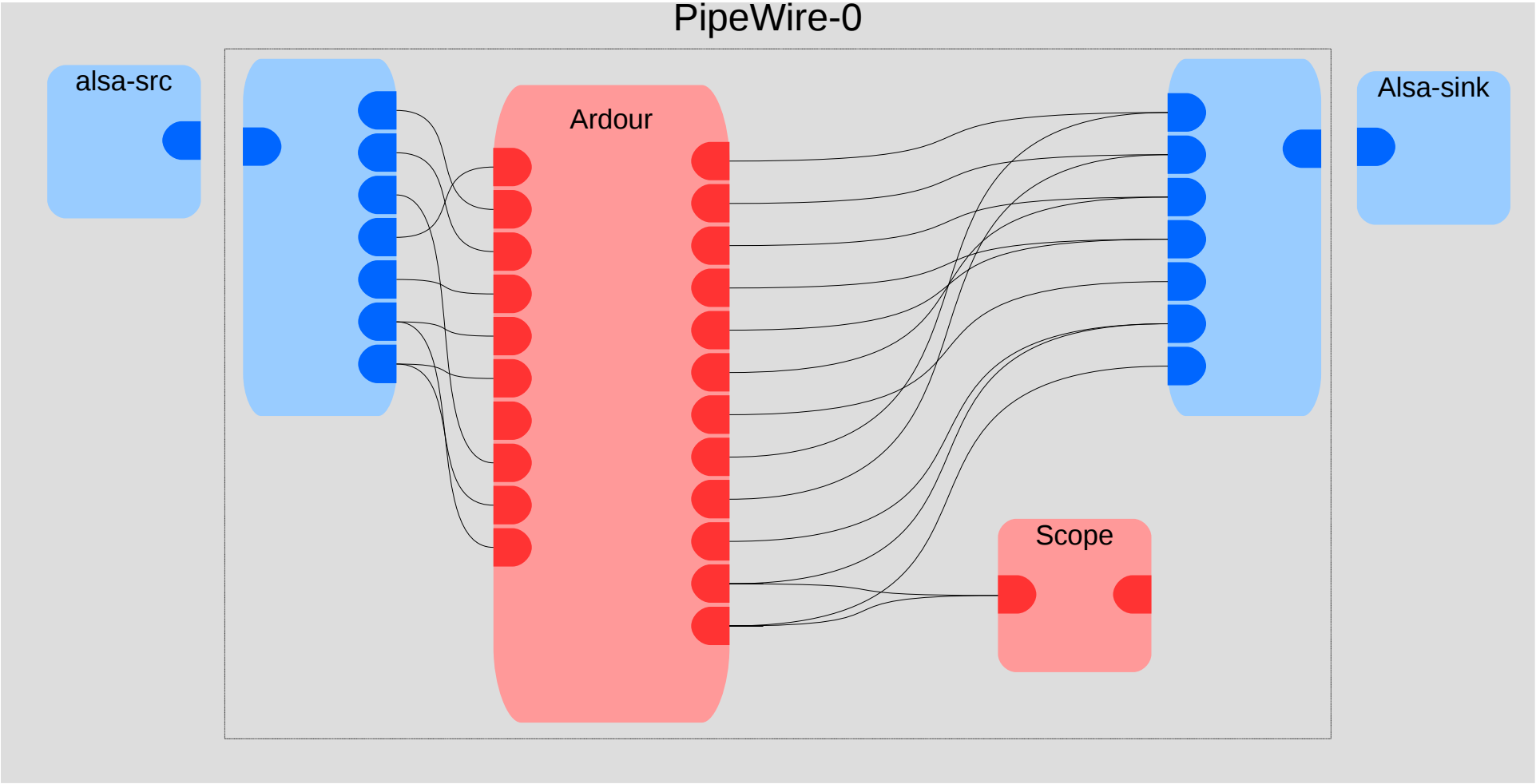
Wayland screen sharing



Audio server



Pro audio

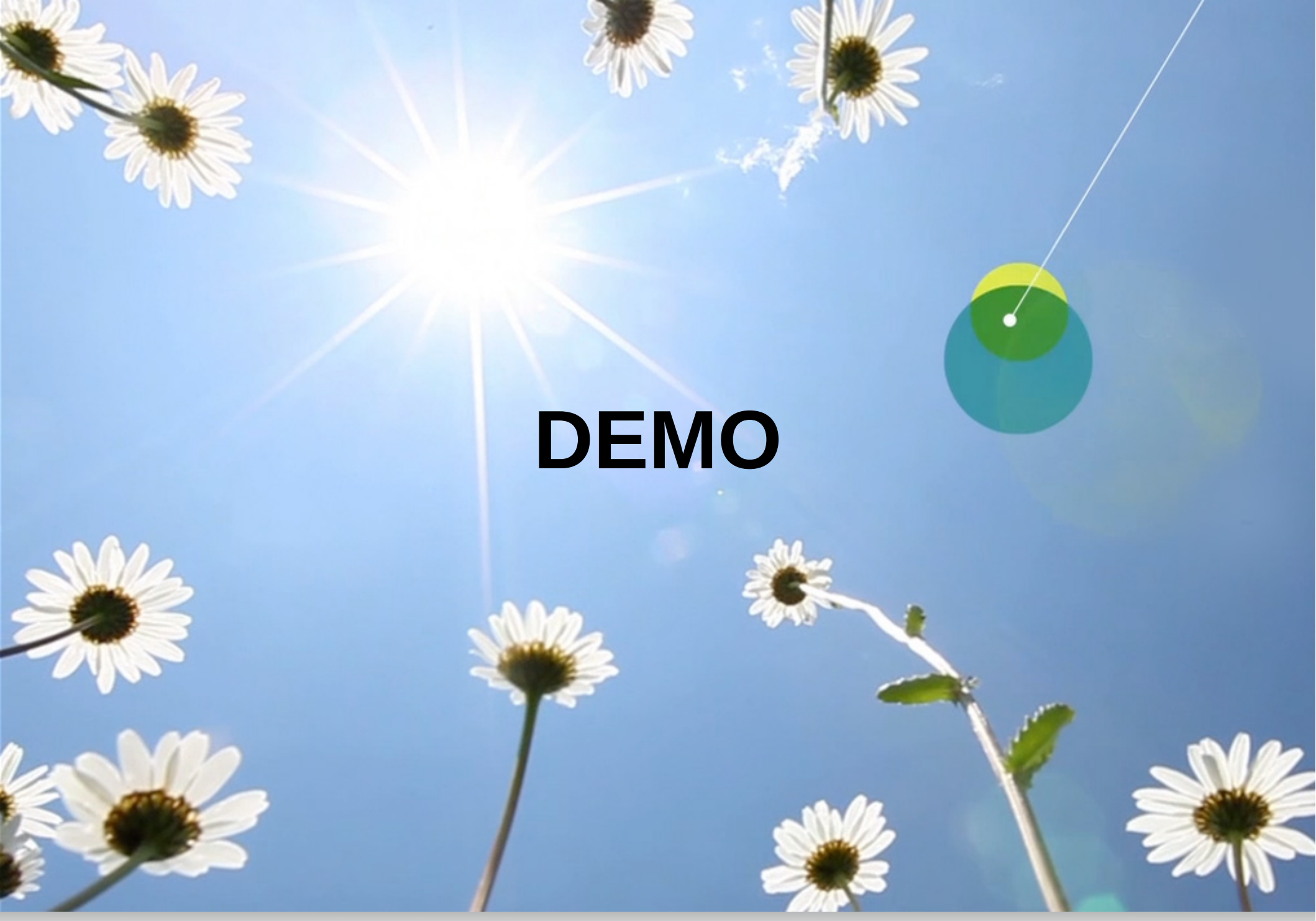


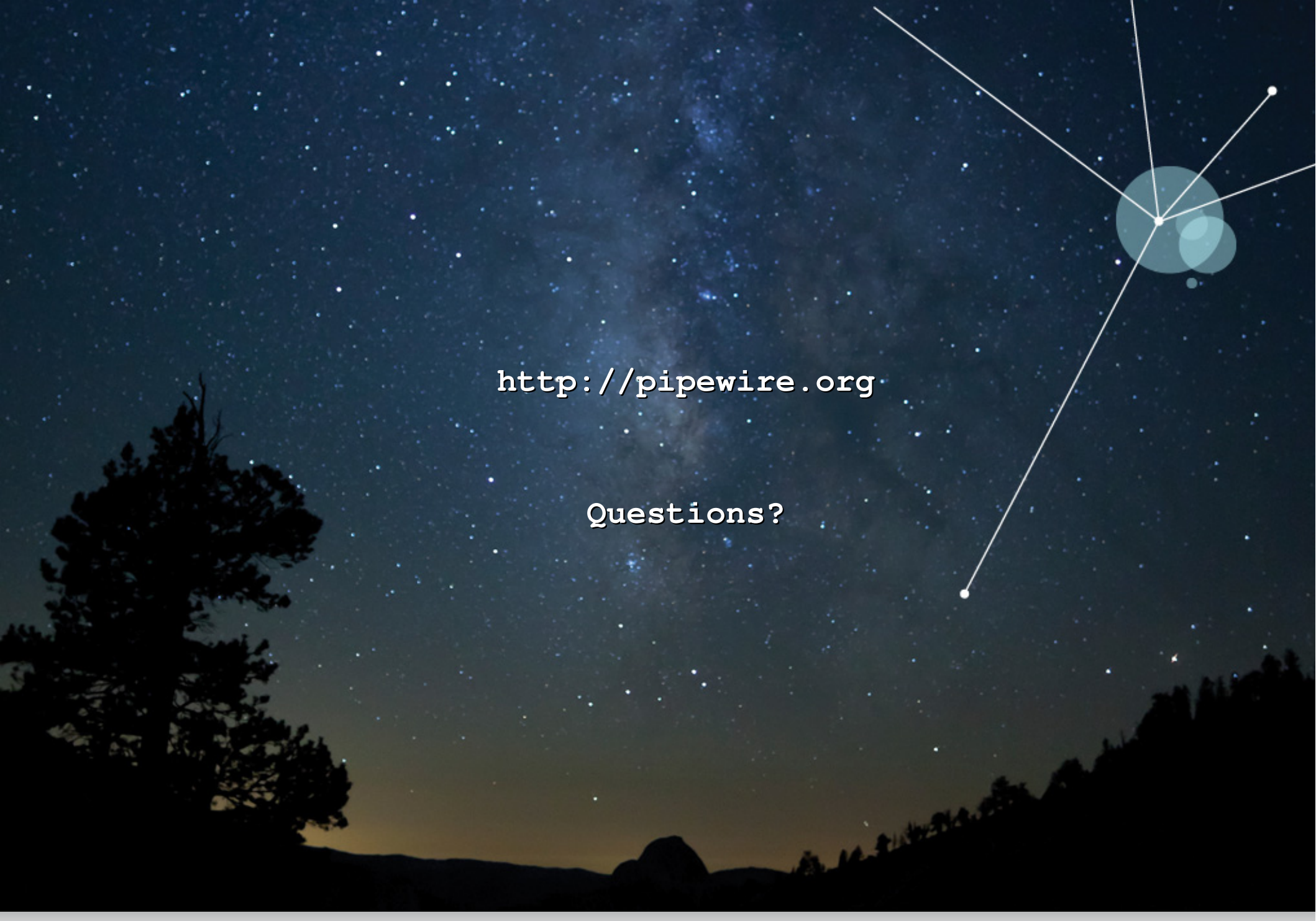
Status

- Dynamically loadable modules, Access control, RT scheduling, Native protocol like wayland + object model, Media upload and download
- V4l2 capture + sharing
- Alsa capture + playback + mixing
- Jack server
- Alsa plugin
- Some tools: monitor, cli
- In Fedora 27

Future plans

- More graph operations
  - Generic scheduler
  - Execution domains, Pro Audio vs consumer
  - Atomic graph updates
- Audio playback/mixing with ringbuffers
- Latency/timing


DEMO


http://pipewire.org
Questions?