



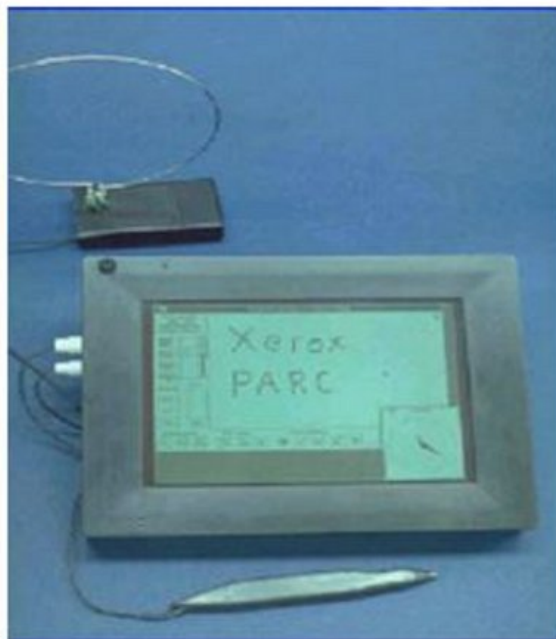
Mirror Hall

Virtual network displays to bridge mobile and desktop

Liveboard



ParcPad



ParcTab

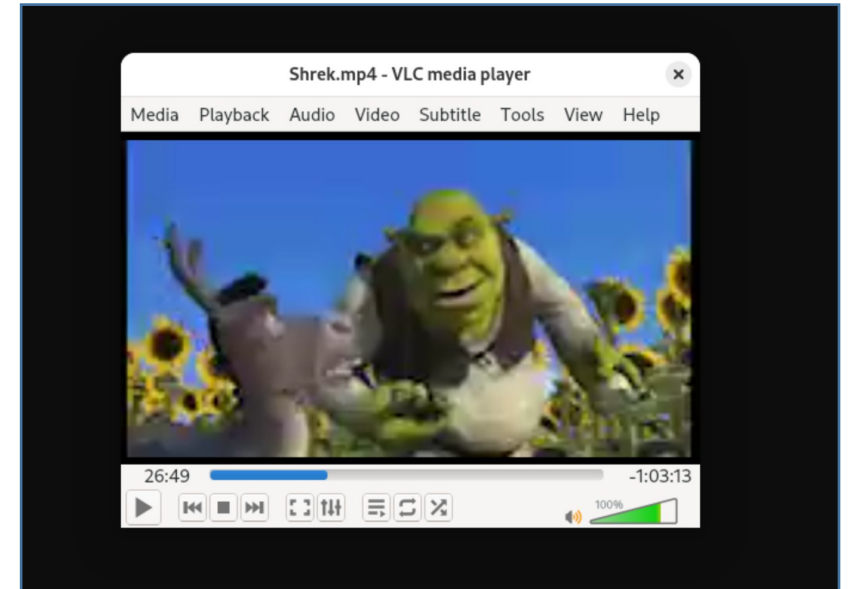
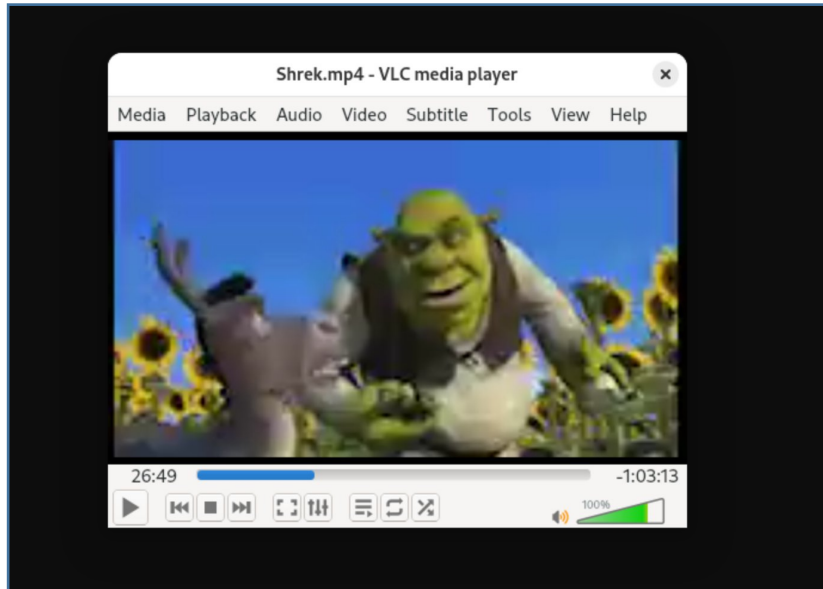


In 1993, Xerox PARC pushed for “ubiquitous computing” [1]

Context

- Why are devices still so hard to interface with each other?
 - E.g., convergence, peer-to-peer file sharing, are still wonky
 - Industry favoured proprietary products over long-lasting protocols
- Some solutions for wireless desktop mirroring exist...
 - Moonlight, Sunshine -> fast and stable, mostly for games
 - GNOME Network Displays is a Chromecast/Miracast sharing tool
- We miss an open solution for *virtual* desktop mirroring
 - i.e., extending your screen on another device

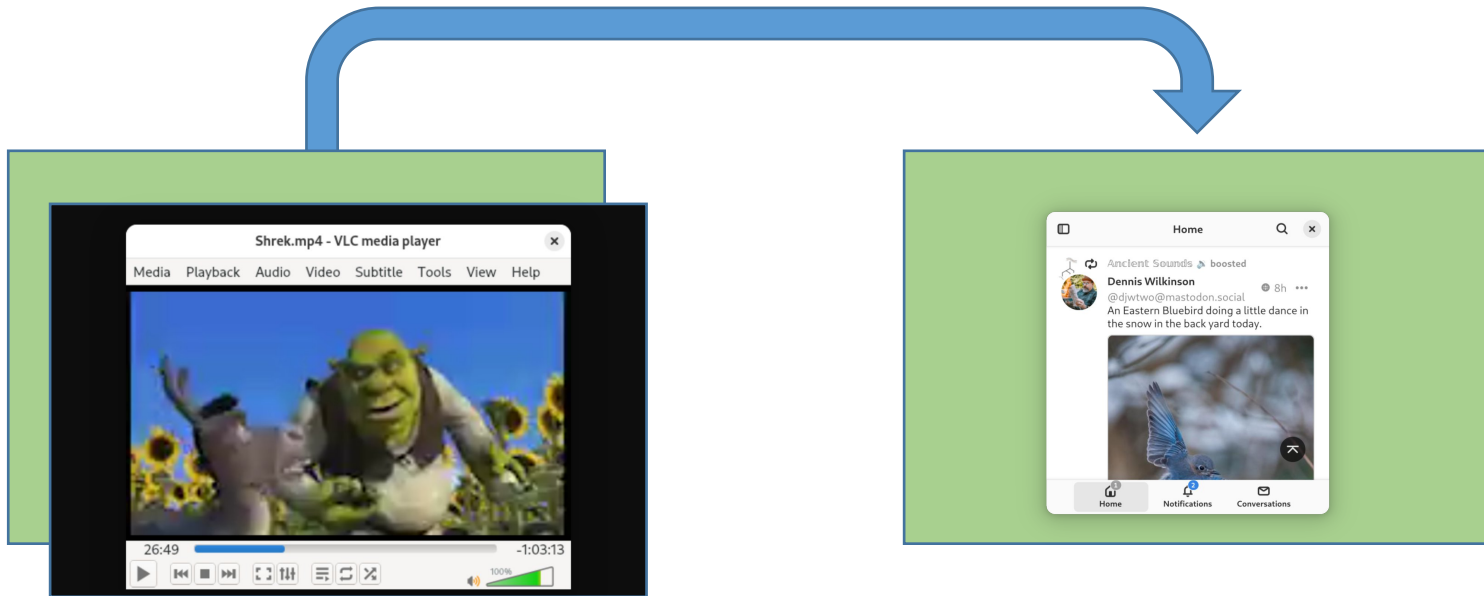
Simple mirroring



chromecast ...

Records the primary screen, and **replicates** it on another.
Lots of good solutions on Linux, all using screen recording API.

What about *virtual* mirroring?



This is possible! It requires spawning a virtual (headless) display, if the WM or graphical stack cooperate.

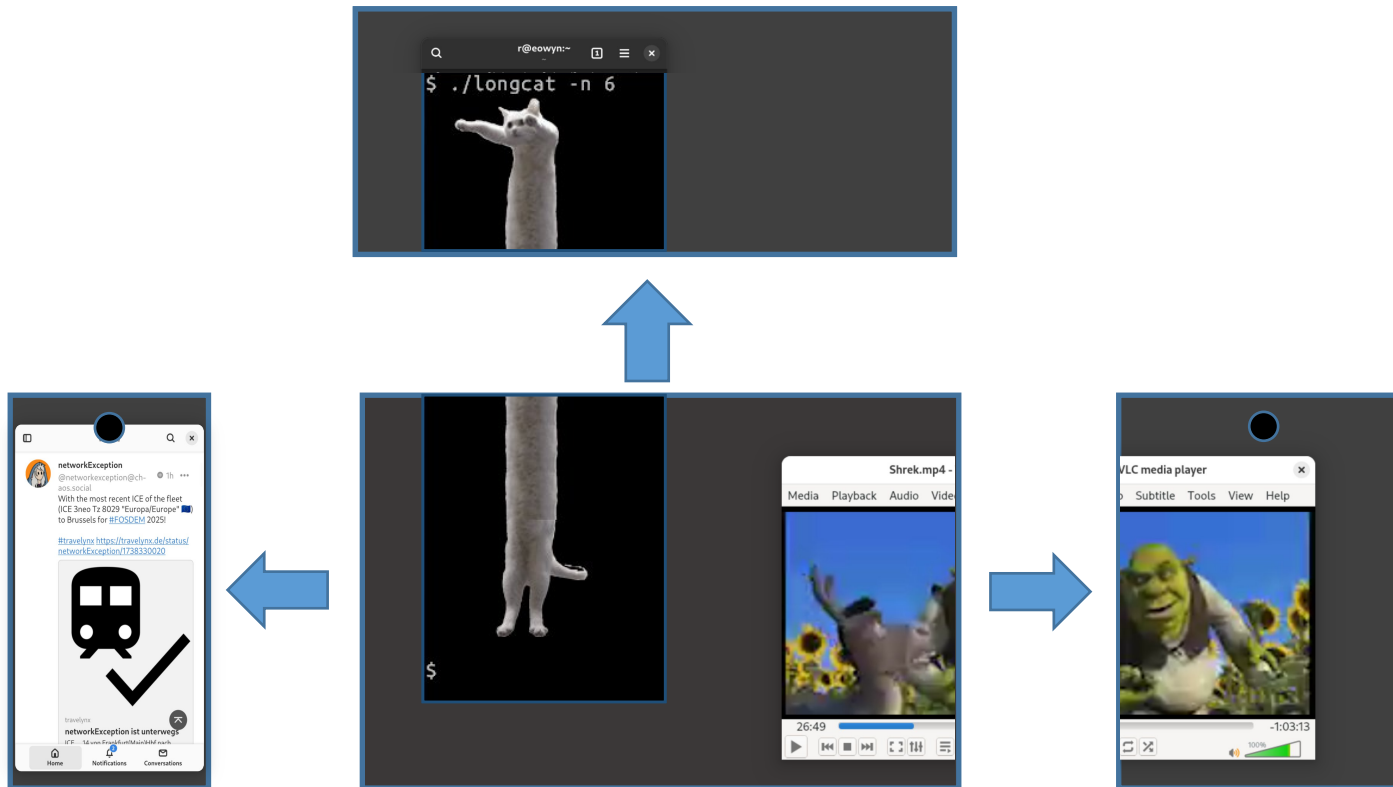


Apple Sidecar allows to use some iPads as extended screens for macOS [2]

Existing solutions wouldn't work

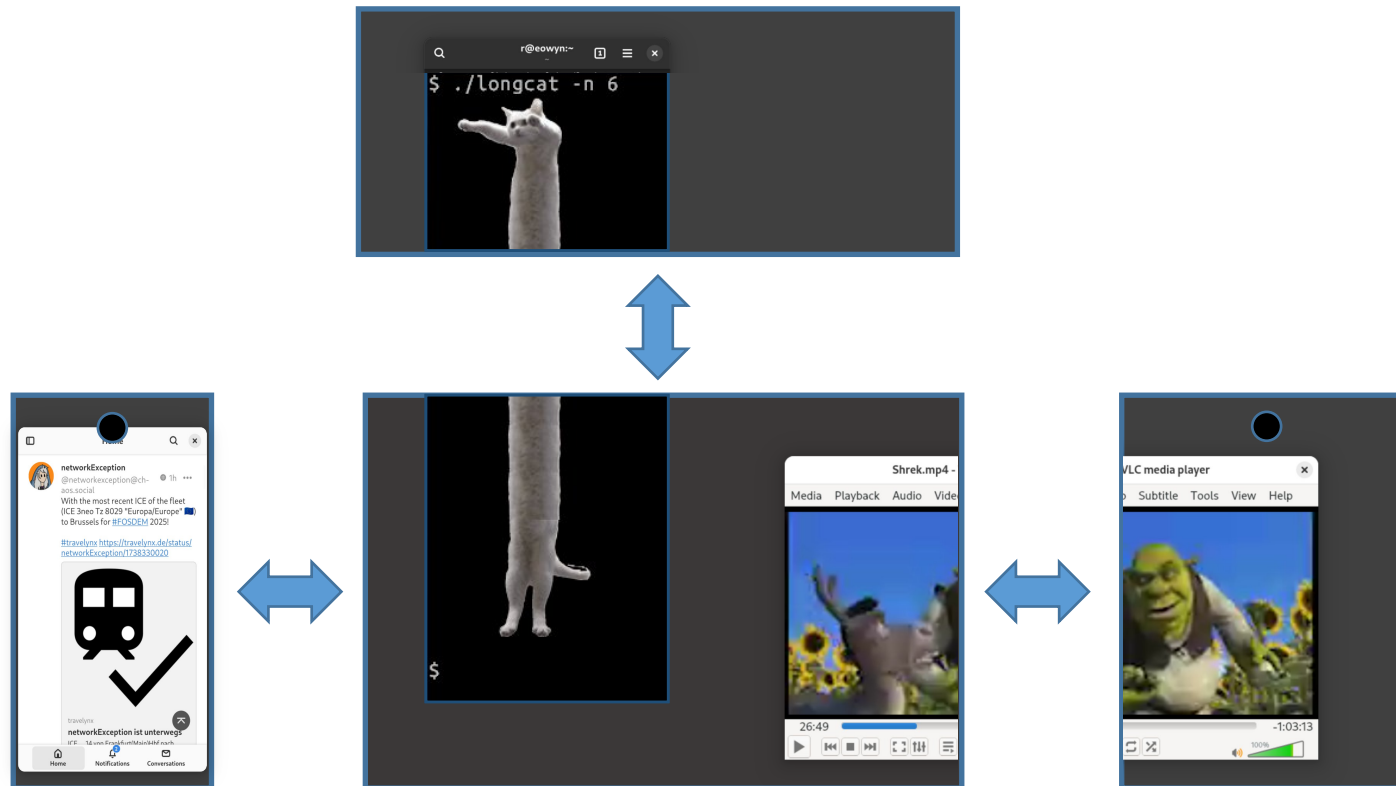
- It's a maze of proprietary protocols :(ul> - Miracast, Chromecast, AirPlay, Sidecar, DisplayLink, ... —> similar core idea
- Existing wireless display standards have high latency – usually ~1s!
 - Mostly TCP-based -> optimized for stability over speed (e.g. for video playback)
 - Implementations are often software-encoded
- All solutions are “unidirectional”: streamers only
 - Turning a Linux device into a Miracast/Chromecast sink is hell
 - Requires e.g. firmware or NetworkManager patches for ad-hoc (Wi-Fi direct)
- Usually meant for one stream at a time

Virtual mirroring, but *multiplied*?



Still possible, but some **conditions apply**: availability of video buffers, hardware encoding buffers, etc.

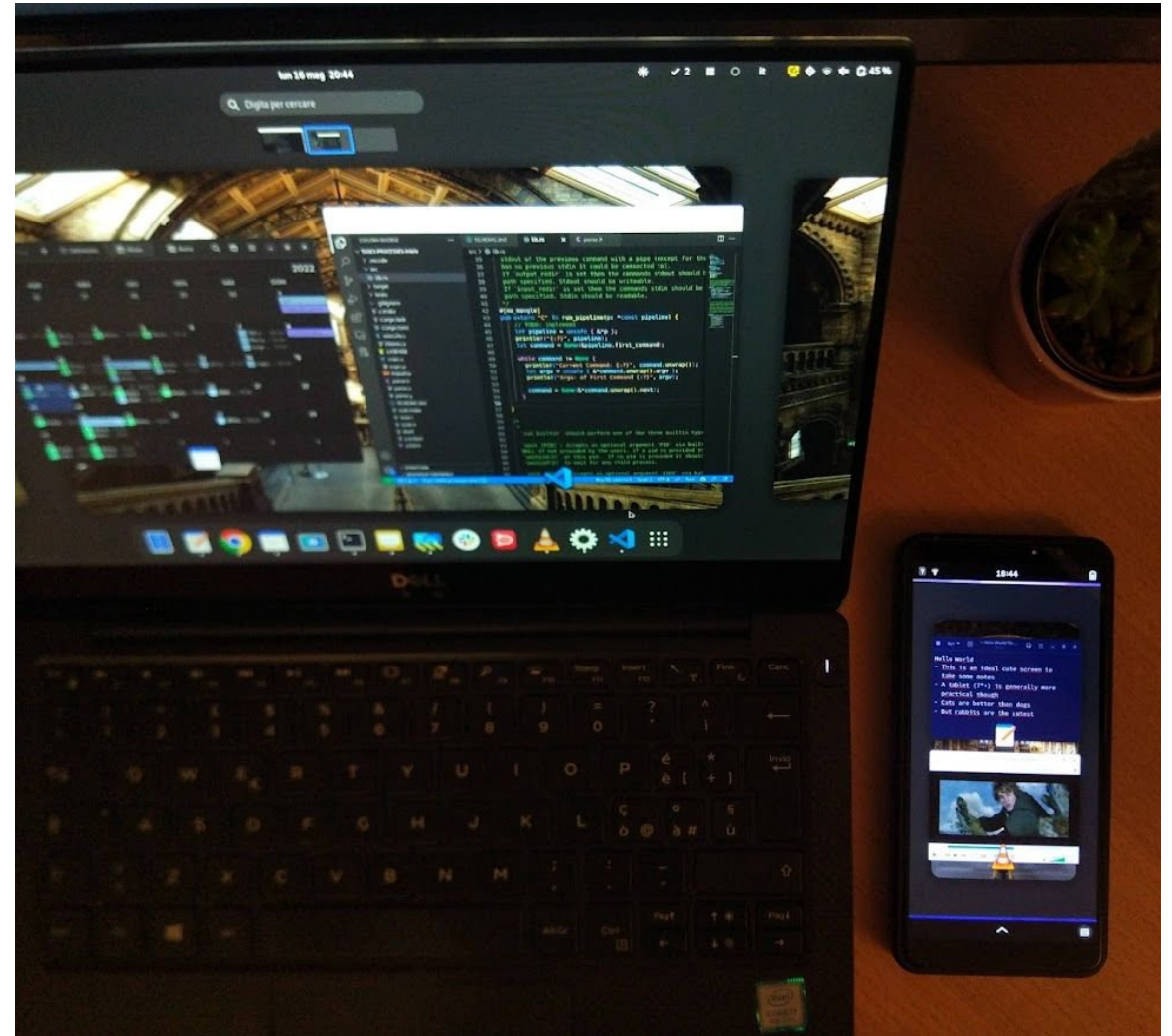
...multiple *bidirectional* virtual mirrors?



Each device should be able to **choose its role**, as a streamer or receiver, at runtime.

Timeline, so far

- 2020: GNOME 40 introduces “Headless native backend and virtual monitors”
 - https://gitlab.gnome.org/GNOME/mutter/-/merge_requests/1698
- May, 2022, first prototype: “Using a Linux phone as a secondary monitor”
 - <https://tuxphones.com/howto-linux-as-second-wireless-display-for-linux/>
- Nov 2023: first Mirror Hall beta
 - <https://fosstodon.org/@tuxdevices/111454321215302030>
- Sep 24: First release!
 - “Mirror Hall: peer-to-peer screen sharing between Linux devices”
<https://notes.nokun.eu/post/2024-09-22-mirrorhall/>



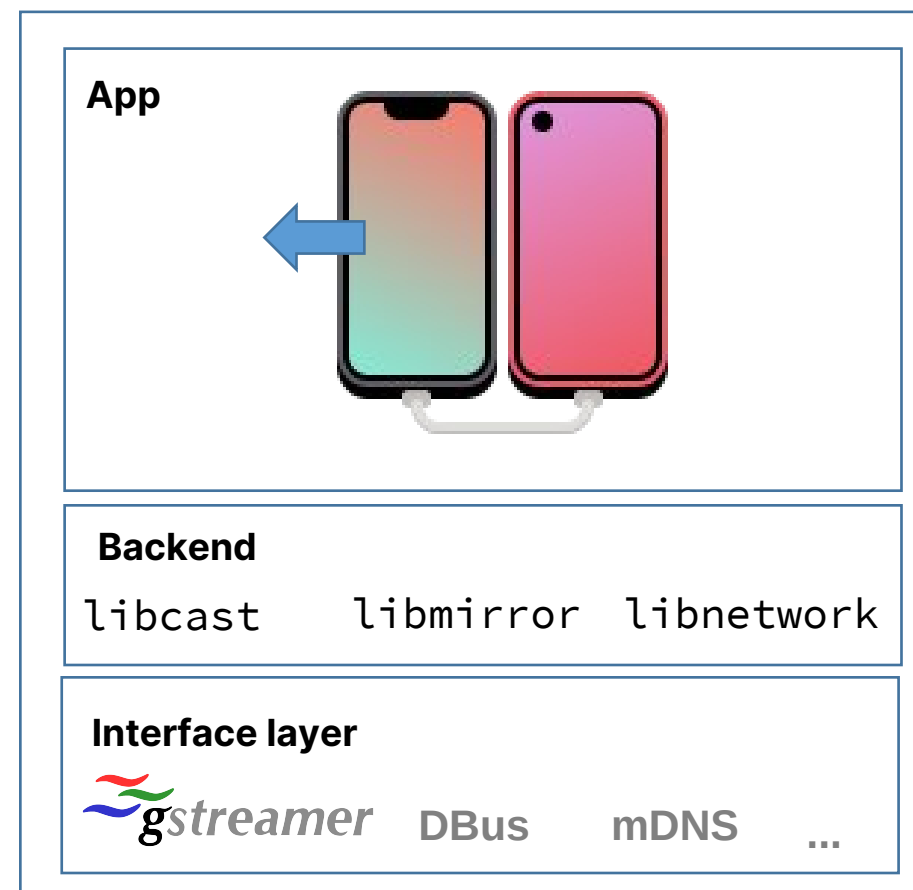
(Mirror Hall window)

How?

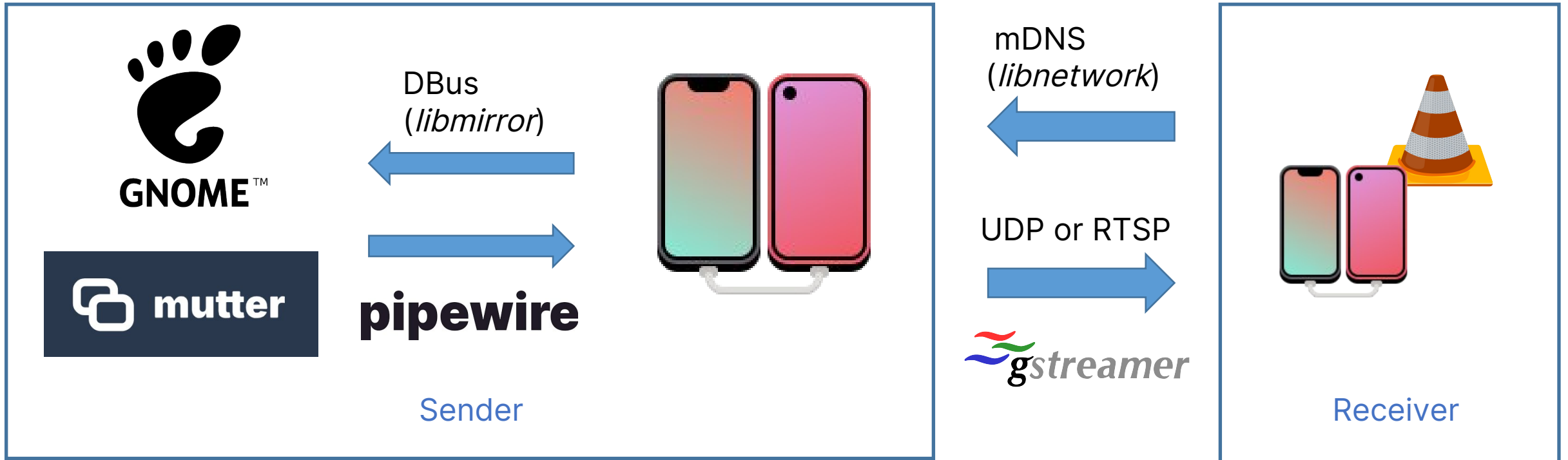


Flatpak

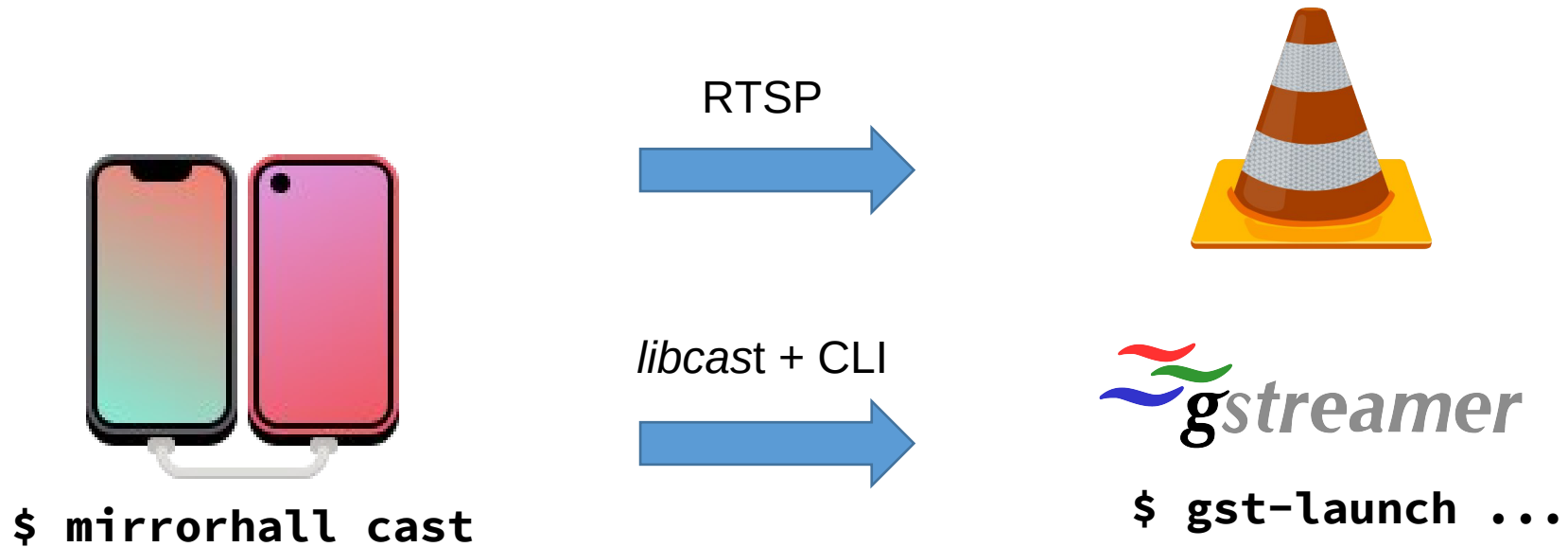
- **MirrorHall** — player/streamer app
 - Adw+Gtk4+custom widgets (player, etc.)
- `libmirror` — creates virtual displays
 - Detects desktop environment
 - Uses D-Bus (for now) to create virtual sink and record it using a PipeWire handle
- `libcast` — streams video over the network
 - Generates the fastest pipelines for video streaming using available hardware accelerators on the host (i.e. GPUs)
 - Intel (vaapi), Qualcomm (venus), Broadcom...
 - Handles network transmission also via GStreamer
- `libnetwork` — handles network communication
 - Advertises Mirror Hall instance on local networks using mDNS (a bit like Chromecast/Airplay/...)
 - Keeps track of health of stream (WiP)
 - Note: the **video** network stream (UDP) is not handled by libnetwork, but offloaded to libcast/gstreamer directly



How?



Bonus: Retrocompatible Mirroring



so you can use Mirror Hall (somewhat painfully) on clients without Mirror Hall installed

How to *build* Up a



Pipeline



How To: Virtual sinks

- We won't talk about mDNS, UDP, health check, etc.
 - ...sort of boring
- *libmirror* only supports Mutter
 - Other DEs don't expose virtual sink APIs yet?
 - Expanding *libmirror* to support them should be quite easy!
- Demo scripts:
 - <https://gitlab.com/tuxphones/side-displays>
 - <https://gist.github.com/louiecaulfield/8688a4dfe59d4f6ec30038be693f7ccf>

How To: Virtual sinks

- No interaction with Wayland / kernel: **Mutter** can create a virtual backend for us
- Set of D-Bus calls to Mutter's screencast APIs:
 - **/org/gnome/Mutter/ScreenCast** -> **CreateSession**
-> Returns stream name: `'/org/gnome/Mutter/ScreenCast/Session/u{x}'`
 - **/org/gnome/Mutter/ScreenCast/Session/u{x}** -> **RecordVirtual ({})**
-> Returns stream path: `'/org/gnome/Mutter/ScreenCast/Stream/u{x}'`
 - **Start()** -> call after to start the session on the stream
 - **/org/gnome/Mutter/ScreenCast/Stream/u{x}**
 - **Create listener on signal: PipeWireStreamAdded** -> get PipeWire stream ID
`$ dbus-monitor --session "type='signal',interface='org.gnome.Mutter.ScreenCast.Stream'"`
- Then call **Start ()** on session...
 - > PipeWire stream will show up in the event handler!

D-Spy ☰

Session **System**

Bus Names ▼

- ca.desrt.dconf
Activatable: Yes, PID: 3205
- ca.desrt.dconf-editor
Activatable: Yes
- com.belmoussaoui...
Activatable: Yes
- com.feralinteractiv...
Activatable: Yes
- com.github.rafostar...
Activatable: Yes
- com.uploadedlobst...
Activatable: Yes
- de.haeckerfelix.Fra...
Activatable: Yes
- io.bassi.Amberol
Activatable: Yes
- io.posidon.Paper
Activatable: Yes
- io.posidon.Paper.Se...
Activatable: Yes
- io.snapcraft.Launcher

org.gnome.Mutter.ScreenCast ✕

Bus Address unix:path=/run/flatpak/bus
Name org.gnome.Mutter.ScreenCast
Owner :1.24
Process ID 3108

Object Path

- ▶ org.freedesktop.DBus.Peer
- ▼ org.gnome.Mutter.ScreenCast.Stream
 - ▼ **Properties**
Parameters ↪ Vardict (read-only)
 - ▼ **Signals**
PipeWireStreamAdded (uint32 node_id)
 - ▼ **Methods**
Start () ↪ ()
Stop () ↪ ()
- ▼ /org/gnome/Mutter/ScreenCast/Stream/u5
 - ▼ **Interfaces**
 - ▶ org.freedesktop.DBus.Properties
 - ▶ org.freedesktop.DBus.Introspectable

Object Path /org/gnome/Mutter/ScreenCast/Stream/u4
Interface org.gnome.Mutter.ScreenCast.Stream
Method Start
Parameters

Result

Elapsed Time ∅: 0.0030 Min: 0.0007 Max: 0.0285

Execute
Copy

How To: Streaming

- Once we have the PipeWire stream object, we can send it over the network via GStreamer:
 - The source (**pipewiresrc**) feeds the incoming stream to the GStreamer pipeline
 - The encoder (**x264, libav,openh264...**) transforms the stream into the desired format (H264, H265, VP8, ...)
 - The payload-encoder (**rtph264pay**) segments the stream into transmittable packets
 - The sink (**udpsink, rtspsink, autovideosink**) plays the result or transmits it over the network
 - Queues add buffering (and improve stability)

```
$ gst-launch-1.0 pipewiresrc path=110
! video/x-raw,width=1280,height=720
! queue
! x264enc
! tune=zerolatency bitrate=6500
! speed-preset=faster ! queue
! rtph264pay
! udpsink host=a.b.c.d port=6906
```

Or, to simply play the PipeWire stream we obtained...

```
$ gst-launch-1.0 pipewiresrc path=110
! video/x-raw,width=1280,height=720
! videoconvert ! queue
! autovideosink
```

How To: Receiving

The other way around...

```
r@eowyn ~> mirrorhall sink 1234
```

```
Mirrorhall CLI - version 0.1.1
```

```
Tip: run with GST_DEBUG=3 for debugging output
```

```
Sink started on port 1234 - press Ctrl+C to stop
```

```
Best decoder: avdec_h264
```

```
Tip: You can use this command to replicate the pipeline outside of Mirror Hall.
```

```
$ gst-launch-1.0
```

```
    udpsrc port=1234 ! queue !           <- Accept UDP stream on port 1234  
    rtph264depay ! queue !             <- Extract video from RTP packet  
    avdec_h264    ! queue !             <- Decode H.264 using best decoder  
    [... convert / parse ... ]        <- Not required for auto-converted pipeline  
    videoconvert !                    <- Auto-convert to a player-accepted format  
    autovideosink                       <- Play video using any available UI sink
```

All About That Latency

- *libcast* is an accelerated pipeline generator
 - Basically a database that generates a compatible pipeline for devices using Qualcomm (ARM) venus, Intel/AMD (VAAPI), or known encoders like libav, openh264, and x264 with custom profiles
 - Should prioritize zero-copy / stateless in the future
- UDP vs. TCP
 - Latency is *considerably* lower at the expense of video artifacts when connection degrades
- Cap on stream quality (FPS and resolution)
 - We cannot control video quality “live” using UDP
- Try to do minimal encoding work
 - Tweak X264, openh264, libav profiles for simplicity
 - Optimized for **speed over precision** (i.e., temporary artifacts may appear if the connection is weak)



Limitations

- MirrorHall requires UDP traffic to flow on ports 6900 to 6999
 - The port is randomized to allow multiple instances on the same host
 - Currently, hole-punching via Flatpak is not really an option

iptables:

```
$ sudo iptables -A INPUT -p udp --dport 6900:6999 -j ACCEPT  
$ sudo iptables -A OUTPUT -p udp --sport 6900:6999 -j ACCEPT
```

firewalld (Fedora):

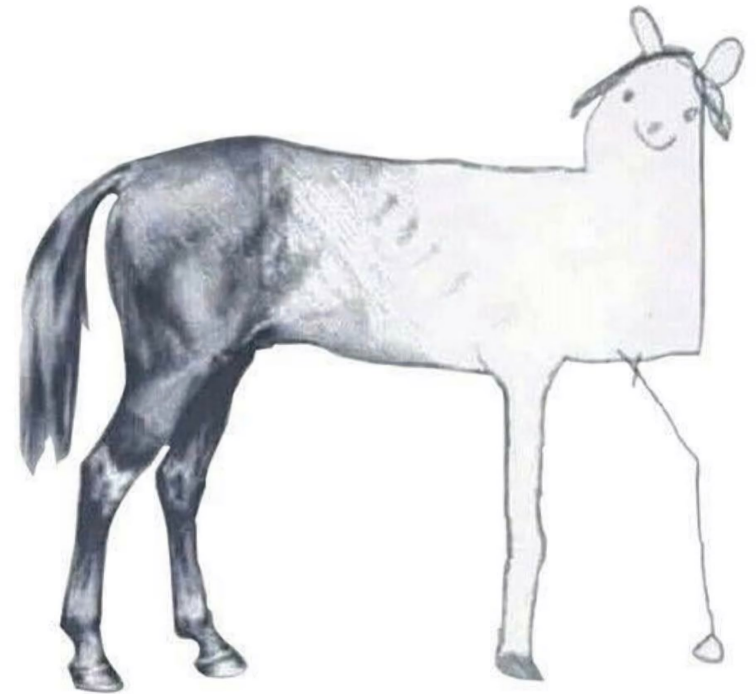
```
$ sudo firewall-cmd --permanent --add-port=6900-6999/udp  
$ sudo firewall-cmd --reload
```

ufw (Ubuntu):

```
$ sudo ufw allow 6900:6999/udp
```

nftables (postmarketOS):

```
$ sudo nft add rule inet filter input udp dport 6900-6999 accept  
$ sudo nft add rule inet filter output udp sport 6900-6999 accept
```



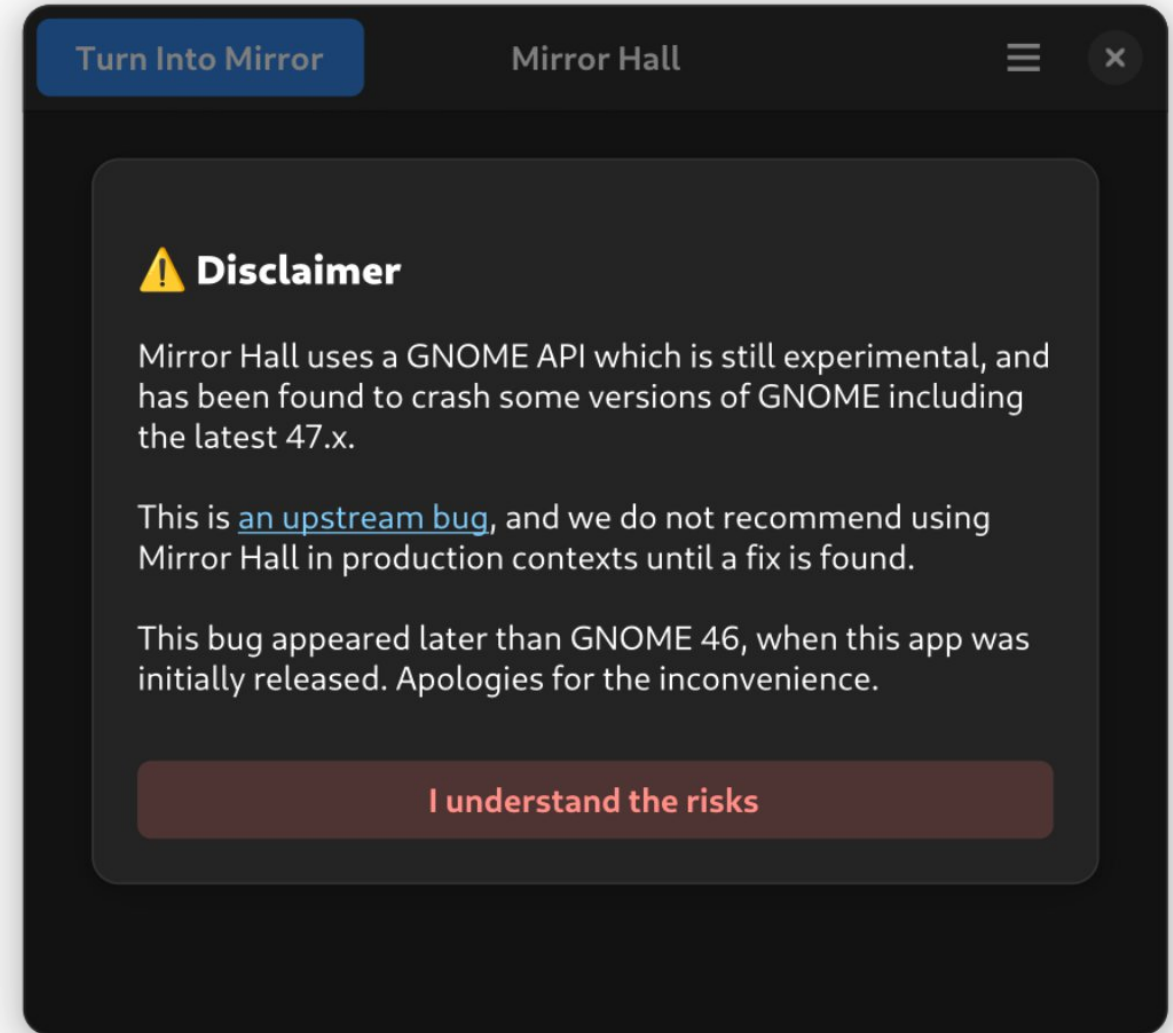
Limitations pt. 2

- Newer versions of GNOME **crash** when closing Mirror Hall - bad release timing :D
 - Mutter 43.x, 47.0, 47.1 are affected
 - 47.2 *seems* to work fine so far
 - Try at your own risk :)
- Flathub's version is slightly slower as it does not include proprietary encoders
- Some other crashes esp. on ARM
 - 0.1.1 fixed some — testers needed!

```
r@eowyn:~$ gnome-shell --version
GNOME Shell 47.2
```

★★★★★ Doesn't work..?

21 October 2024



Next steps

- **Encryption** and stability
 - Insecure raw UDP H.264 for now
 - No existing solution within GStreamer (?)
 - Split up app and protocol layers, adding UDP hole-punching
- > Ideally: Rust + iroh
- Add mirroring of input methods
 - e.g., for the use case of signing a document using Wacom-enabled tablet
 - Maybe: proxy input events directly?
 - Even Faster
 - Zero-copy + stateless components

iroh

less net work for networks

ids 88k chat 131 online YouTube License MIT

[Docs Site](#) | [Rust Docs](#)

Conclusion

- We're at 0.1.1. There's still a **long** way to go...
 - Divide into smaller components, improve transmission stability, and make it work upstream (i.e., not GNOME-only)
- Thanks to all who supported me!
 - Sonny Piers, Caleb Connolly, Jonas Dressler, Tobias Bernard, Robert Mader, Rafał "rafostar" Dzięgiel, ...
- Looking for collaborators!
 - Extend to other platforms (KDE, Sway) / protocols / hardware (RPi, ...)
- Keep in touch?
 - DM me on Mastodon: @tuxdevices
 - Write me directly — r@nokun.eu
 - Let's meet in Berlin!

Attribution

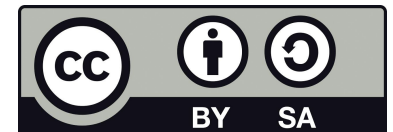
References. (1) Schilit, Adams, et al., "The PARCTAB mobile computing system," Proceedings of IEEE 4th Workshop on Workstation Operating Systems. WWOS-III, Napa, CA, USA, 1993, pp. 34-39

Images. p.2: see (1), p.3: Myrabella / Wikimedia Commons / CC BY-SA 3.0; Apple Sidecar (apple.com/de/newsroom/2019/06/apple-previews-macos-catalina/); p.4: tuxphones.com, p.22: <http://www.supertuxkart.at/page3/page3.html>, p.16: modified from the cover of A. Malm, *How To Blow Up A Pipeline*, Verso Books

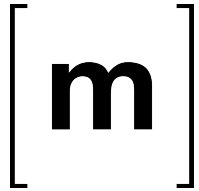
Artwork. The Mirror Hall icon was designed by Tobias Bernard.

This presentation is released under Creative Commons CC-BY-SA 4.0.

Trademarks and artwork belong to their respective owners, which sounds kind of obvious tbh.



More Links + Q&A



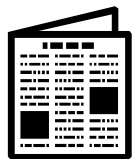
Chat rooms (join us!)

[@mirrorhall:gnome.org](https://matrix.to/#/!mirrorhall:gnome.org) | t.me/MirrorHallApp



Me (email) / TuxPhones (Mastodon 🐼)

@tuxdevices@fosstodon.org | r@nokun.eu



Mirror Hall 0.1.0 — Technical Deep Dive

notes.nokun.eu/post/2024-09-22-mirrorhall/



GitLab — Mirror Hall

gitlab.com/nokun/mirrorhall

