# The Steam Runtime

## Using Debian to run games on Linux

*Simon McVittie, Collabora / Debian*
*MiniDebconf Cambridge, November 2026*



COLLABORA

# So you want games on Linux?

**Well, *I* do**

# History of Valve gaming on Linux

- Steam Client for Linux (2013)
- Steam Machines (2015)
- Proton (2018): Windows games on Linux
- Steam Deck (2022) →
- Steam Machine (2026)
- Steam Frame (2026) →
- FEX-Emu (2026): x86 games on aarch64
- Your computer?





# Why the Steam Runtime?

- Build your game on some distribution. Will it work on some other distribution?
- We need something that game developers can target
- Old game on new OS
  - Gamers expect games to work for years to come
  - … even if they used `libpng.so.12` and `libssl.so.0.9.8`
- New game on old OS, within reason
  - Debian stable, Ubuntu LTS, RHEL, …

# Why the Steam Runtime?

- Build the Steam Client on some distribution. Will it work on another distribution?
- We need something that Valve can target
- Doesn't necessarily need to work forever
  - Steam Client auto-updates itself to latest version
- Does need to work for everyone, within reason



COLLABORA

# Why can't you just?

Oh no

# Use the OS's libraries?

- This is fine if you're part of the OS
  - … and can update on the same timescales
  - Steam isn't, and neither are Steam games
- Is OpenSSL version 1 or 3 or 4?
- Is it `libjpeg.so.62` or `libjpeg.so.8`?
- Will the answers to those questions be different in 5 years?

# Bundle the game's dependencies?

- Some **can't** be frozen at one version: graphics drivers
- You release a game in 2026
- Later, a gamer plays it on their 2036 computer
- In 2026, their GPU hadn't been invented yet
- So we can't possibly bundle drivers that will support it

# Bundle the game's dependencies?

- Some can be bundled, but it doesn't work
- Bundle `libstdc++.so.6` from gcc-14 with your game
- Later, Mesa needs `libstdc++.so.6` from gcc-16
- Older bundled library breaks the graphics driver
- True for any recursive dependency of the graphics driver
  - `libX11`, `libdrm`, `libwayland-client`, …

# Bundle the game's dependencies?

- Some just can't be bundled, like glibc
- `libc.so.6` is tightly coupled to `libdl.so.2`
- `libdl.so.2` is tightly coupled to `ld.so(8)`
  - For example `/lib64/ld-linux-x86-64.so.2`
- `ld.so(8)` absolute path is part of the ABI
  - ELF `PT_INTERP` header of every executable

# Bundle selected dependencies?

- Yes, ish
- Bundle only a high level dependency: can work
  - Low level dependency had better be new enough, otherwise kaboom
- Bundling a low level dependency breaks higher levels
  - Bundled freetype breaks unbundled fontconfig
- We can do this, but it only works at the top of the stack
  - Fine for game-specific libraries that we can't or won't add to the runtime
  - Not suitable for anything the graphics driver might use

# LD_LIBRARY_PATH a mile long?

- We did try this: Steam Runtime 1 'scout', circa 2013
- It works, except for when it doesn't
  - Can't force a newer glibc
    - So we're stuck on the oldest supported OS, or older
  - Need to be very careful to pick the newer version
    - Difficult to achieve with one hand tied behind your back
  - Sometimes host libraries look compatible but aren't
    - `libcurl` upstream vs. Debian

# I know, let's put it in a container

- Solved problem? Docker, Podman, lxd, Incus, …
- But games typically have graphics *[citation needed]*
- Gamers want frames per second, therefore GPUs
  - We're going to want some drivers
  - You'll notice a theme here

# Flatpak: containers for apps

- Builds a graphics driver as an extension for the container
- Special extension for the NVIDIA proprietary driver
  - Because it's constrained to be in lockstep with the kernel side
- Works, to a point
  - Can build the latest Mesa on tolerably old runtimes
  - Older runtimes make this increasingly hard
  - Runtimes on Flathub are only supported for 1-2 years
  - That's not enough, some games haven't been updated since 2014



COLLABORA

# The part where we solve it

This is that part

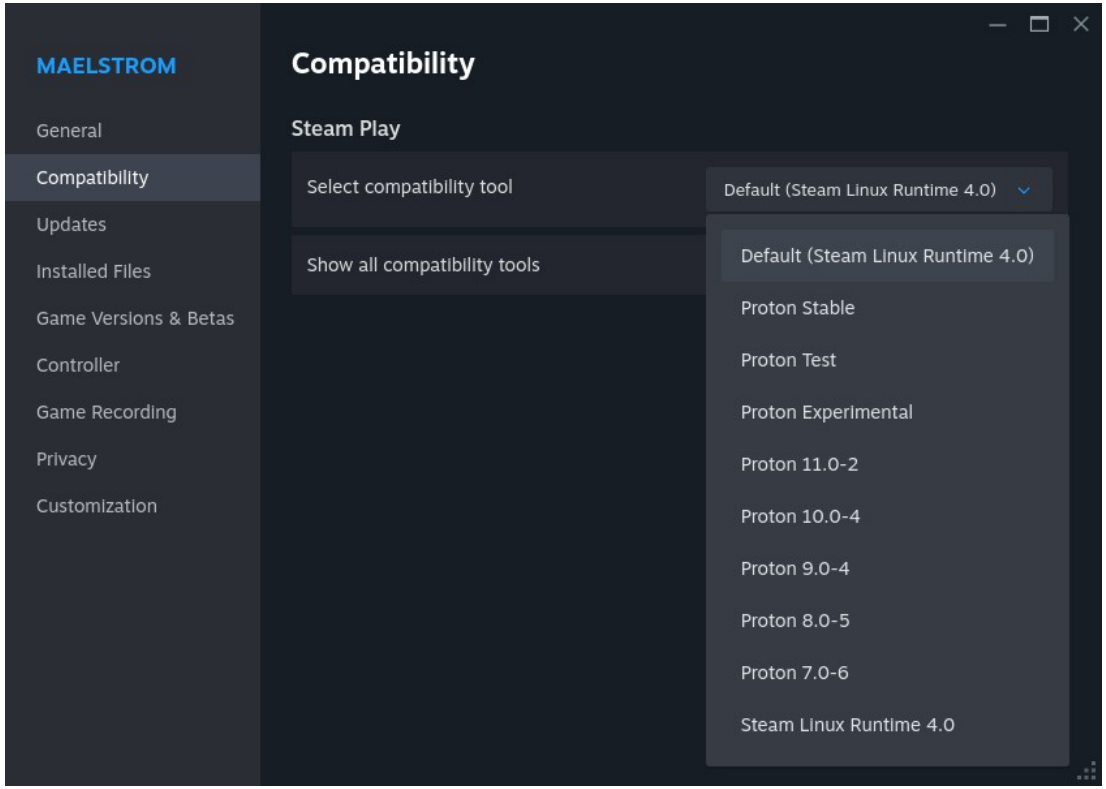
# Steam Runtime SDK

- Build your game in here
  - OCI container: Docker, Podman, Toolbox, Distrobox
  - Tarball for a chroot, if you insist
- Limited or no GPU acceleration
  - Oh well, not usually necessary at compile-time
  - You do get llvmpipe/lavapipe for automated tests
- It's Debian, mostly
  - `apt-get install what-you-need`
  - Backports of selected important libraries (and toolchains)

# Steam Runtime Platform

- SDK subset: run your dedicated server or tests in here
  - OCI container: Docker, Podman, Toolbox, Distrobox
  - Tarball for a chroot, if you must
- Limited or no GPU acceleration
  - Oh well, not usually necessary for dedicated servers
  - You do get llvmpipe/lavapipe for automated tests
- Same package versions as the SDK, but no `-dev`
- "Platform" has the same meaning here as in Flatpak

# Steam Linux Runtime



# Steam Linux Runtime

- Application layer: under our control
  - Exactly the same libraries as the Platform
  - Deleted `dpkg`, `apt`, `perl` … to save space
  - `/usr/lib/*-linux-gnu`
- Graphics drivers: from the host ("graphics provider")
  - The only drivers known to support the user's GPU
  - `/usr/lib/pressure-vessel/overrides/lib/*-linux-gnu`
- Dependencies from the provider or the Platform
  - Whichever is newer, in whichever location is appropriate

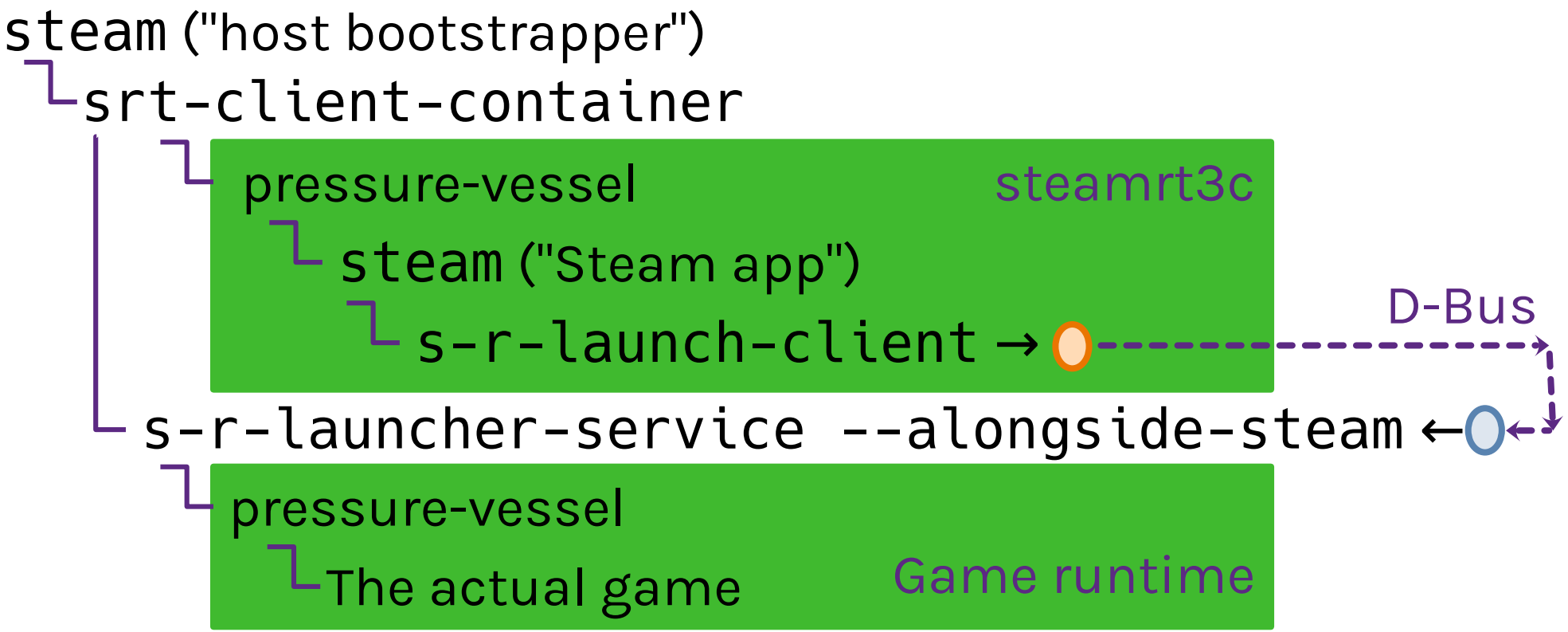
# pressure-vessel

- Steam Linux Runtime's container-runner
- Similar to to Flatpak's sandboxing layer
  - Lots of code reuse from Flatpak, but intentionally not a sandbox
  - Uses bubblewrap for the low-level parts, just like Flatpak
- No package-manager layer
  - That's the Steam Client's job (Steampipe)
- Extra complexity around graphics drivers
  - Here be dragons

# Containerized Steam Client

- 64-bit Steam Client, in beta since October 2026
  - Previously "Experimental SteamRT3 client"
- Mostly runs inside a container
  - `steamrt3c`, a modified version of Steam Runtime 3
- Older 32-bit Steam Client was mostly not in a container
  - … except `steamwebhelper`, already in a container since 2024

# Containerized Steam Client



# Newer games use newer runtimes

- Steam Linux Runtime 4 'steamrt4'
  - Based on Debian 13 (2025)
- Steam Linux Runtime 3 'sniper'
  - Based on Debian 11 (2021), with older ABIs
- Steam Linux Runtime 2 'soldier'
  - Based on Debian 10 (2019), with much older ABIs
- Steam Linux Runtime 1 'scout'
  - Based on Ubuntu 12.04 (2012), older ABIs again







COLLABORA

# It can't be that easy

Well, yes, but actually no

# Steam Linux Runtime 1 'scout'

- Based on Ubuntu 12.04 'precise' (2012)
- Games should ideally have been compiled in its SDK
  - Many game developers didn't – containers were a new thing at the time
  - Accidentally worked in the `LD_LIBRARY_PATH` runtime anyway
- Steam Linux Runtime 1 (original)
  - A strictly scout-based container
  - Games worked if developer was careful (Half-Life, Portal, …)
  - Many didn't, because they made more assumptions



# Steam Linux Runtime 1 'scout', redone

- Launch a Steam Linux Runtime 2 'soldier' container
  - This was a contemporary OS version at the time
  - Now held in stasis for legacy compatibility
- Run the old `LD_LIBRARY_PATH` runtime inside that
  - In practice, works a lot more often than a strict scout container
  - "`scout-on-soldier`" internally
- There's also the Legacy Steam Runtime but don't use that
  - This is just the old `LD_LIBRARY_PATH` runtime. Good luck!
  - Might or might not work depending on OS, desktop, lunar cycle

# Graphics drivers aren't just drivers

- Some Vulkan layers are functionally necessary
  - Mesa device selection
  - NVIDIA Optimus
- Developers and gamers expect other Vulkan layers to work
  - MangoHUD
  - Renderdoc
- Not just Vulkan, either
  - OpenXR layers, EGL external platforms, `LD_PRELOAD` modules…

# So which library is newer?

- Drivers, layers, etc. have dependencies
  - We already said we need provider or container, whichever is newer
- In an ideal world, this would be simple
- We do not live in that world

# Difficulty: Easy

- `libSDL3.so.0`, a symlink to…
- `libSDL3.so.0.4.8`
  - `Version: 3.4.8`
- Every compatible change raises minor or micro version
- SONAME bump if and only if ABI breaks
- Compare versions, choose the higher one, like `ldconfig`
- `libelf.so.1` → `libelf-0.192.so` is the same idea

# Difficulty: Normal

- `libstdc++.so.6`
  - Versioned symbols, grouped by ELF "symver"
- Public symbols explicitly exported
  - `std::bad_exception::what() const@GLIBCXX_3.4.9`
- List symvers, whichever side has more is newer
  - If some symvers are only in Platform and some only on host: problem!
- Can do the same with individual symbols
  - libxcb family

# Difficulty: New Game +

- `_Fork@GLIBC_2.34`
  - Is ABI, introduced in 2.34, obviously
- `_IO_enable_locks@GLIBC_PRIVATE`
  - As the symver suggests, this isn't public ABI
  - Ignore that, only look at the public symbols
- Can do the same with namespaced symbols
  - `libdrm_radeon.so.1` has `radeon_*`: public, `bof_*`: private

# Difficulty: Unfair

- `libedit.so.2` reimplements the `readline` API/ABI
- `append_history@Base 3.1-20180525-0`
  - Is part of the ABI despite not having any special prefix
- `ct_decode_string@Base 3.1-20160618-0`
  - Is not part of the ABI despite not having any obvious indicator
- We have to care about this because Mesa uses libLLVM
  - … and libLLVM uses libedit
  - Could it perhaps … not?

# Library knowledge

```
- libdrm_radeon.so.1:
    compare_by:
      - name
      - symbols
    public_symbols:
      - '!bof_*'
      - 'radeon_*'
      - '!'
      - '*'
```

```
- libedit.so.2:
    check_loadable: false
    from_host_system: true
    generate_dependency: false
    compare_by:
      - symbols
    public_symbols:
      - '!__ct*'
      - '!c_*'
      - '!ce_*'
      - '!ch_*'
      - '!vis'
      - '!wcsdup'
      - '_el_fn_complete'
      - '_el_fn_sh_complete'
      - '_rl_*'
      - 'add_history'
      - 'append_history'
      - 'clear_history'
      - 'completion_matches'
      - 'current_history'
      - 'el_*'
      - 'emacs_*'
      - 'filename_completion_function'
      - 'fn_*'
      - 'free_history_entry'
      - 'get_history_event'
      - '!parse_*'
      - 'unstifle_history'
      - 'username_completion_function'
      - 'using_history'
      - 'where_history'
      - 'write_history'
      - '!'
      - '!*'
```

# How to make the container?

- We need to tell the game to use the graphics-stack libraries
- glibc will often come from the host
  - We don't control, or know, its search path
- At first we tried setting `LD_LIBRARY_PATH`
  - Worked at first, but it turns out that a lot of games overwrite that
- How do normal programs *actually* find their libraries?
  - The answer is `/etc/ld.so.cache`, built by `ldconfig`
  - (except when the path is different because of reasons)

# How to make the container

- Copy the base runtime
  - Use hard links to not waste space
- Populate extra library directory with graphics stack
  - `/usr/lib/pressure-vessel/overrides/lib/*-linux-gnu`
  - Symlinks pointing into `/run/host`
- Delete libraries that the graphics stack has provided
  - This is why we had to copy the runtime
  - Otherwise `ldconfig` gets confused
- Generate a new `ld.so.cache` combining library directories

# How to make the container

- Graphics-stack libraries also have associated data
  - glibc locales, plugins, executables, `ld.so(8)`
  - NVIDIA driver data
  - libdrm quirks data
- Need to get those into the container, too
  - Mostly symlinks pointing into `/run/host`
  - A few special cases use a bind-mount
  - This is another reason why we have to copy-and-modify



COLLABORA

# Lessons learned

Keep on trying ('til you run out of cake)

# Multiarch lets us generalize

- FHS/Fedora wants 32-bit libraries in `/usr/lib`
- Arch wants 64-bit libraries in `/usr/lib`
  - But those two layouts can't both have what they want
- Everyone agrees what's in `/usr/lib/x86_64-linux-gnu`
  - In the cases where it exists at all (Debian family, freedesktop.org SDK)
- Also great for cross-compiling and emulation
- Can we encourage other distros to be multiarch?
  - UAPI team hopes so

# Container setup isn't ideal

- Copying the base runtime is slow if latency is high
  - NFS has high throughput, but high latency for metadata ops
- Can we solve this with `overlayfs`?
  - Maybe, but that could cause exciting new bugs

# Distro-specific quirks are a tradeoff

- We've had to implement many per-distro special cases
- I'm not going to say every divergence is bad
  - Multiarch is amazing
- But every divergence creates work for someone else
  - And creates bugs for our users
- "Upstream first", where feasible
- If not: is this divergence a net benefit?

# Avoid distro-specific ABIs if possible

- ncurses has versioned symbols, but not always
  - Yes on Arch and Debian, no on Fedora
- curl SONAME was historically inconsistent in Debian
  - Particularly affects Steam Runtime 1 'scout'
- Debian added versioned symbols to curl before upstream
- Easy to get a situation where neither "side" can be happy
- Please talk to upstream! They're the coordination point

# Avoid distro-specific ABIs if possible

- We like it when distros package releases
- We run into trouble when distros package snapshots
- If there's no suitable release, ask upstream for one?
- If the library isn't stable, perhaps don't link dynamically?

# Reduce drivers' dependencies

- Static linking where appropriate
  - Avoid expanding the ABI by using visibility or `-Bsymbolic`
  - Steam Runtime's Mesa does this, maybe Debian's could too?
- `dlopen` non-essential things instead of linking directly
  - Extra-important in `LD_PRELOAD` modules, like MangoHUD
  - Can locate them in `/run/host` with `dlinfo( )`, like MangoHUD
  - `RTLD_LOCAL` and `RTLD_DEEPBIND` help to avoid collisions
- Move non-core functionality out of the way
  - libLLVM successfully avoided a dependency on libcurl
  - Perhaps the same can happen for libedit?

# Dependency libraries: control your ABI

- `-fvisibility=hidden` is your friend
  - Side benefit: PE DLLs on Windows always work more like this
- Versioned symbols
  - They don't have to be complicated! `gcc -Wl,--default-symver`
  - Avoid private ABI, or group it into an obviously private symver
- Normal library versioning
  - Semantic versions or something a lot like them
  - Reflect the semantic version in the library filename
  - Some good examples: GLib (Meson), SDL 3 (CMake)



# Thank you!

# Further reading

- https://github.com/ValveSoftware/steam-runtime
- https://github.com/ValveSoftware/steam-for-linux
- https://gitlab.steamos.cloud/steamrt/steam-runtime-tools
- https://gitlab.steamos.cloud/steamrt/steamrt



# We are hiring

## col.la/careers