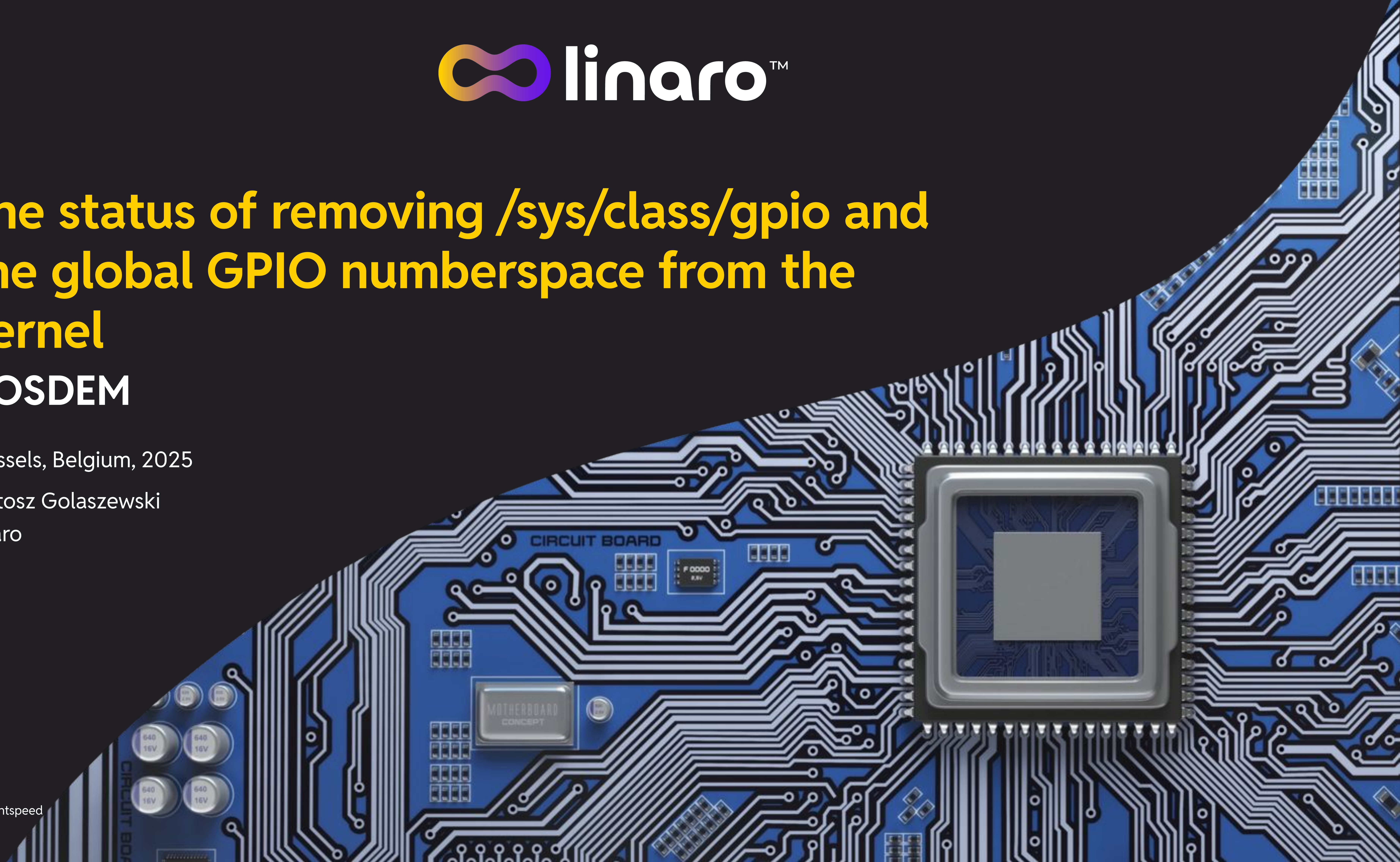
The status of removing /sys/class/gpio and the global GPIO numberspace from the kernel

FOSDEM

Brussels, Belgium, 2025

Bartosz Golaszewski
Linaro



About me

- Linux kernel developer for the Qualcomm Landing Team at Linaro
- 15 years of embedded linux experience
- Maintainer of the GPIO subsystem
- Author and maintainer of libgpiod
- Open-source contributor to many other projects

linaro™

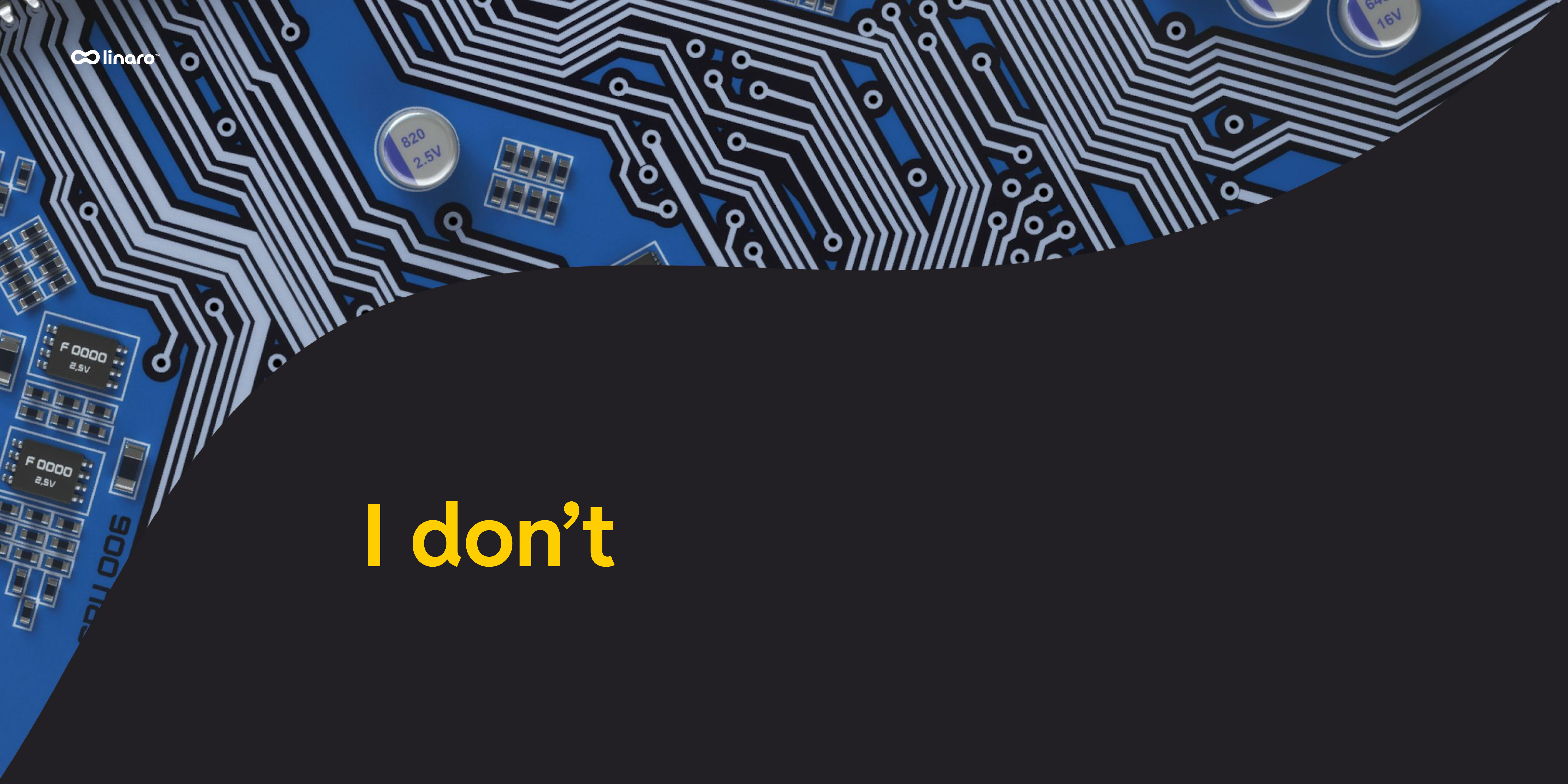
is the software engine of the arm Ecosystem

Linaro empowers rapid product deployment within the dynamic arm Ecosystem.

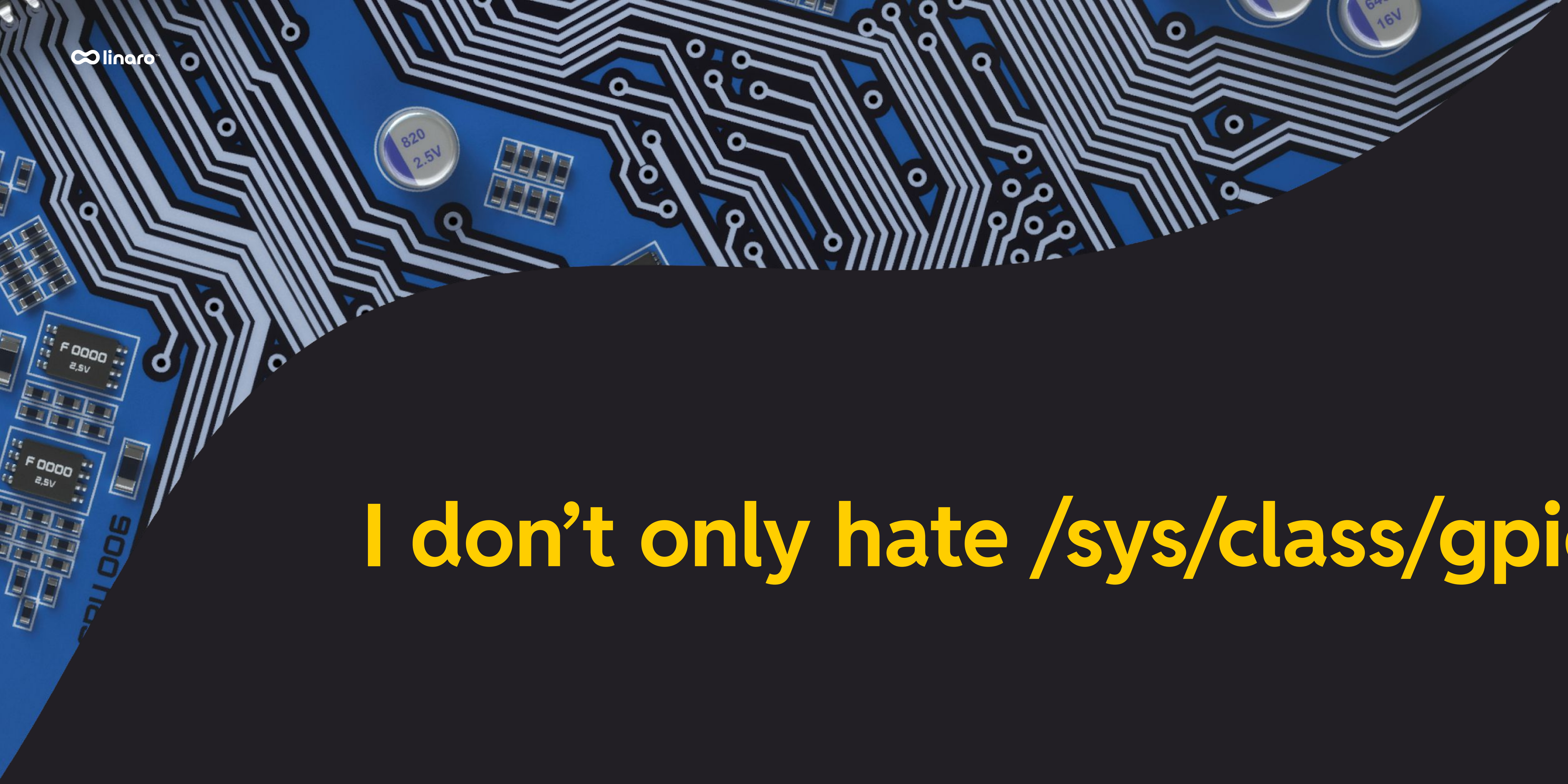
- Our cutting-edge solutions, services and collaborative platforms facilitate the swift **development, testing, and delivery of arm-based innovations,** enabling businesses to stay ahead in today's competitive technology landscape.
- **Linaro** fosters an environment of collaboration, standardization and optimization among businesses and **open source ecosystems to accelerate the deployment of arm-based products and technologies** along with representing a pivotal role in open source discovery and adoption.
- **Automotive, Testing, Linux Kernel, Security, Cloud & Edge Computing, IoT & Embedded, AI, CI/CD, Toolchain, Virtualization**

Linaro has enabled trust, quality and collaboration since 2010

Bart, why do you hate /sys/class/gpio?



I don’t



I don’t only hate /sys/class/gpio

GPIOLIB has a problem with legacy cruft

- **Relevant talk:**
  - *“Compound Interest - Dealing with Two Decades of Technical Debt in Embedded Linux”*
  - `https://www.youtube.com/watch?v=BR41Yg69c9Y`

GPIOLIB has a problem with legacy cruft

- Relevant talk:
  - *"Compound Interest - Dealing with Two Decades of Technical Debt in Embedded Linux"*
  - `https://www.youtube.com/watch?v=BR41Yg69c9Y`
- The biggest issue is having two intertwined ways of keeping track of GPIOs
  - Modern descriptor-based, two-level (chip, line) hierarchy
  - Legacy global GPIO numberspace

Why remove global GPIO numberspace?

- Unify the in-kernel GPIO interfaces
- Use the interface which doesn't allow buggy drivers to claim GPIOs that aren't theirs
- Drop hardcoded GPIO base
- Don't depend on predefined magic values for GPIOs (in kernel and user-space)
- Reduce maintenance burden

What stands in the way?

- Some drivers still don’t use descriptors
  - That’s not a hard problem
  - In-tree drivers can be converted one-by-one
  - We don’t care about breaking out-of-tree drivers
  - It’s just tedious

What stands in the way?

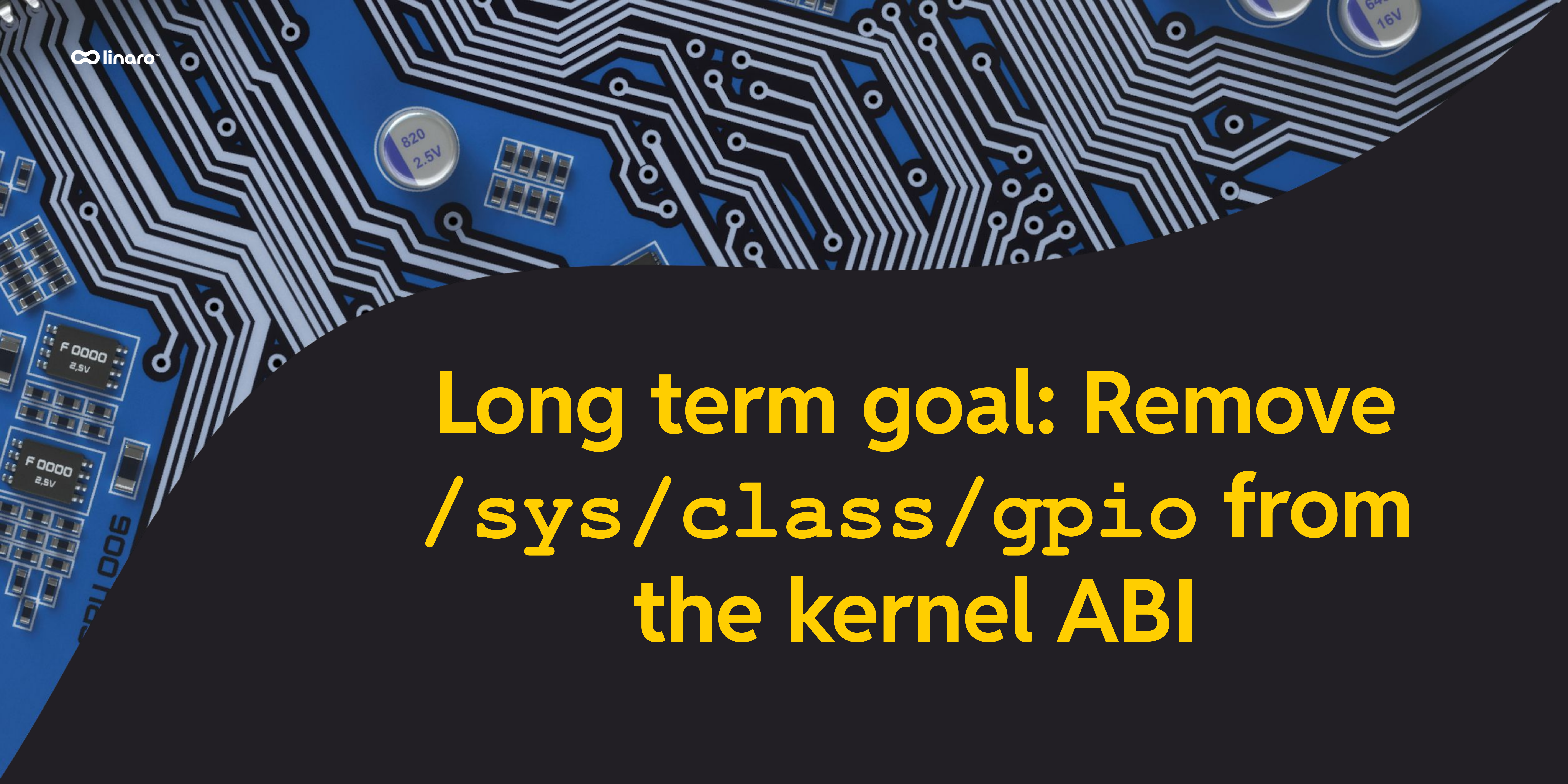
- Some drivers still don’t use descriptors
    - That’s not a hard problem
    - In-tree drivers can be converted one-by-one
    - We don’t care about breaking out-of-tree drivers
    - It’s just tedious
- /sys/class/gpio is a major user of the legacy in-kernel interface
    - This is a hard problem due to advertised uABI stability

`/sys/class/gpio` has issues

- Users rely on brittle shell scripts toggling GPIOs identified by magic numbers
- Implements a rather wonky polling mechanism
- Lacks a lot of features of the character device
- Processes using GPIOs can get in each-other’s way
- The ABI has been inconsistent for 10 years and nobody even noticed

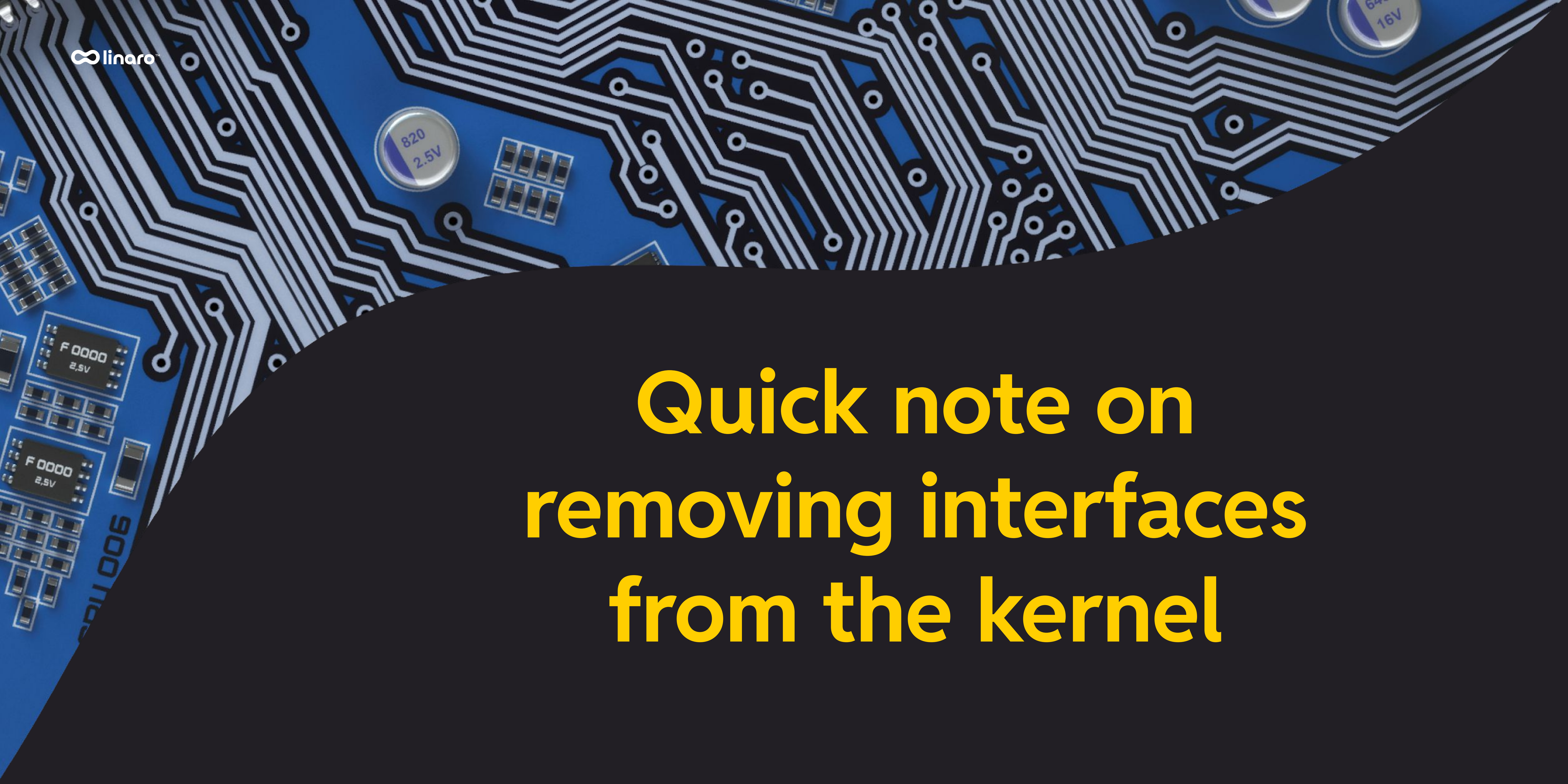
`/sys/class/gpio` also some pros too

- Fine-grained permission control using the VFS ops
- Effectively works as an in-kernel GPIO daemon



Long term goal: Remove `/sys/class/gpio` from the kernel ABI

Prerequisite: Users must stop using it first



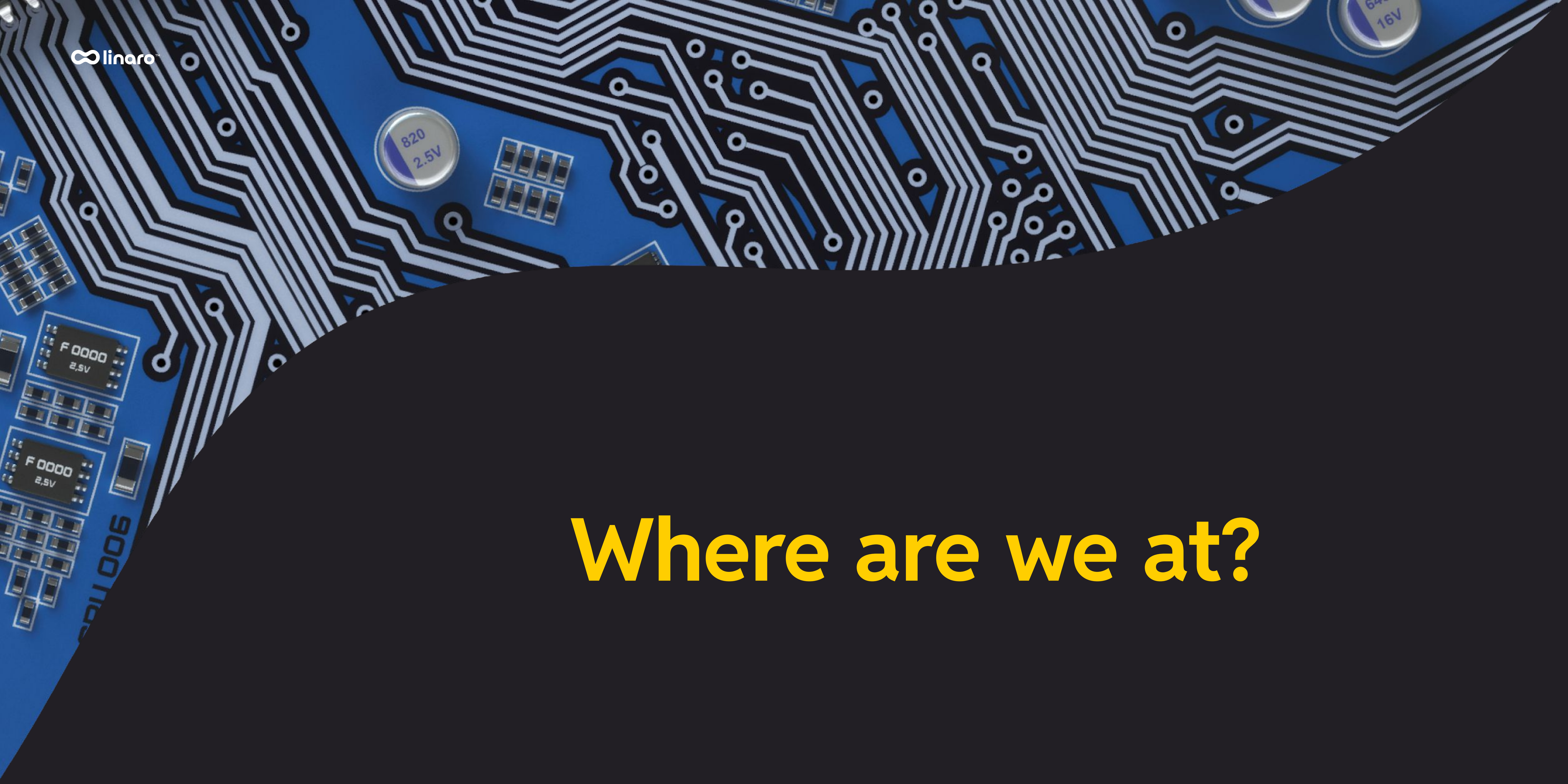
Quick note on removing interfaces from the kernel

It’s not without precedent

- `sysctl()` system call -> removed in linux v5.5
- `/dev/kmem -> removed in linux v5.9`
- `/dev/raw -> removed in linux v5.14`
- Some sysfs classes were dropped over the time
  - `/sys/class/misc/rtc`

But...

- For most part: if user-space objects to backward incompatible changes, we must not remove existing interfaces
- Unless an interface is proven to be harmful
- Which is not the case here :(

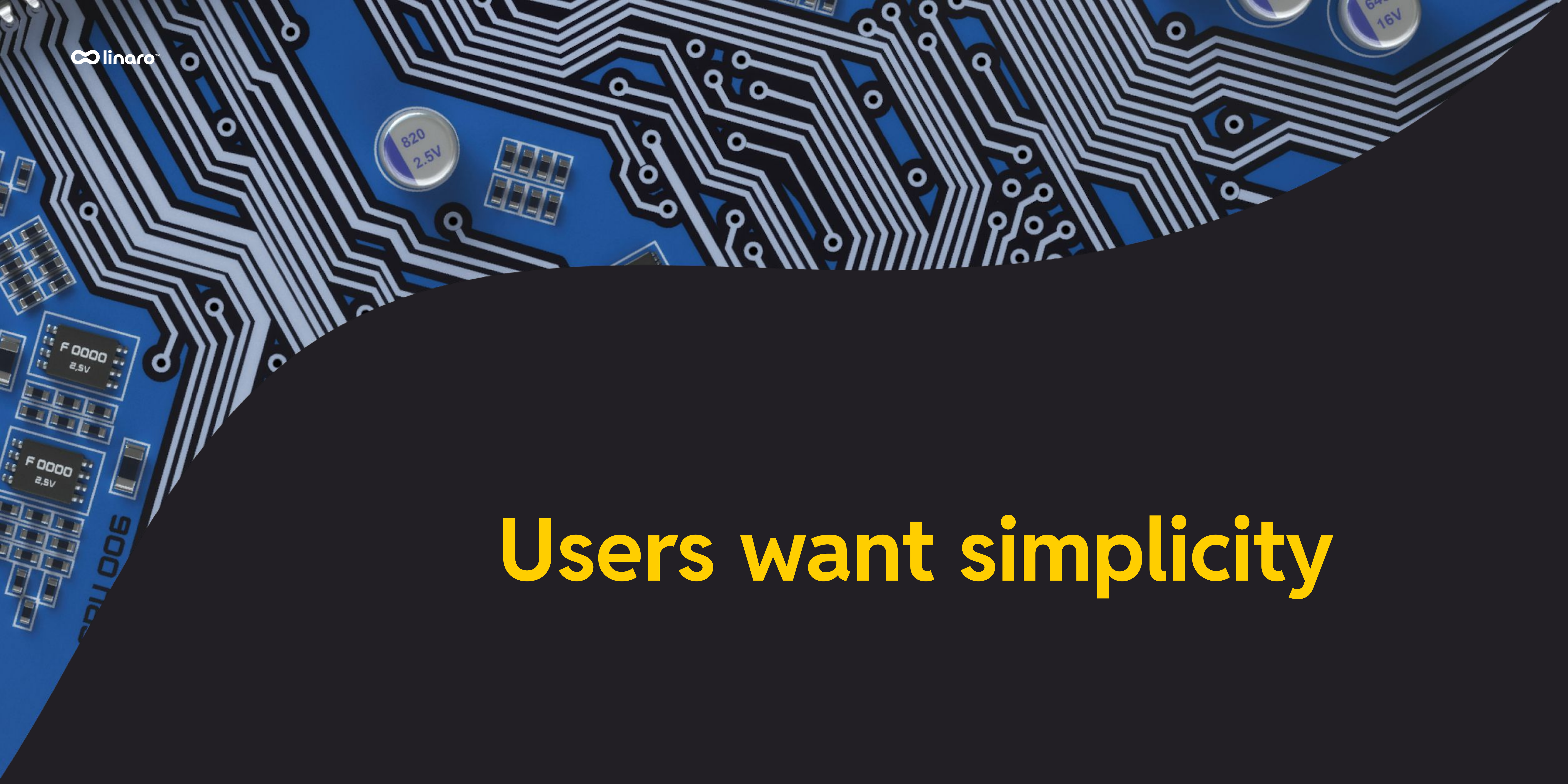


Where are we at?

Nowhere near :(

Proposed alternatives

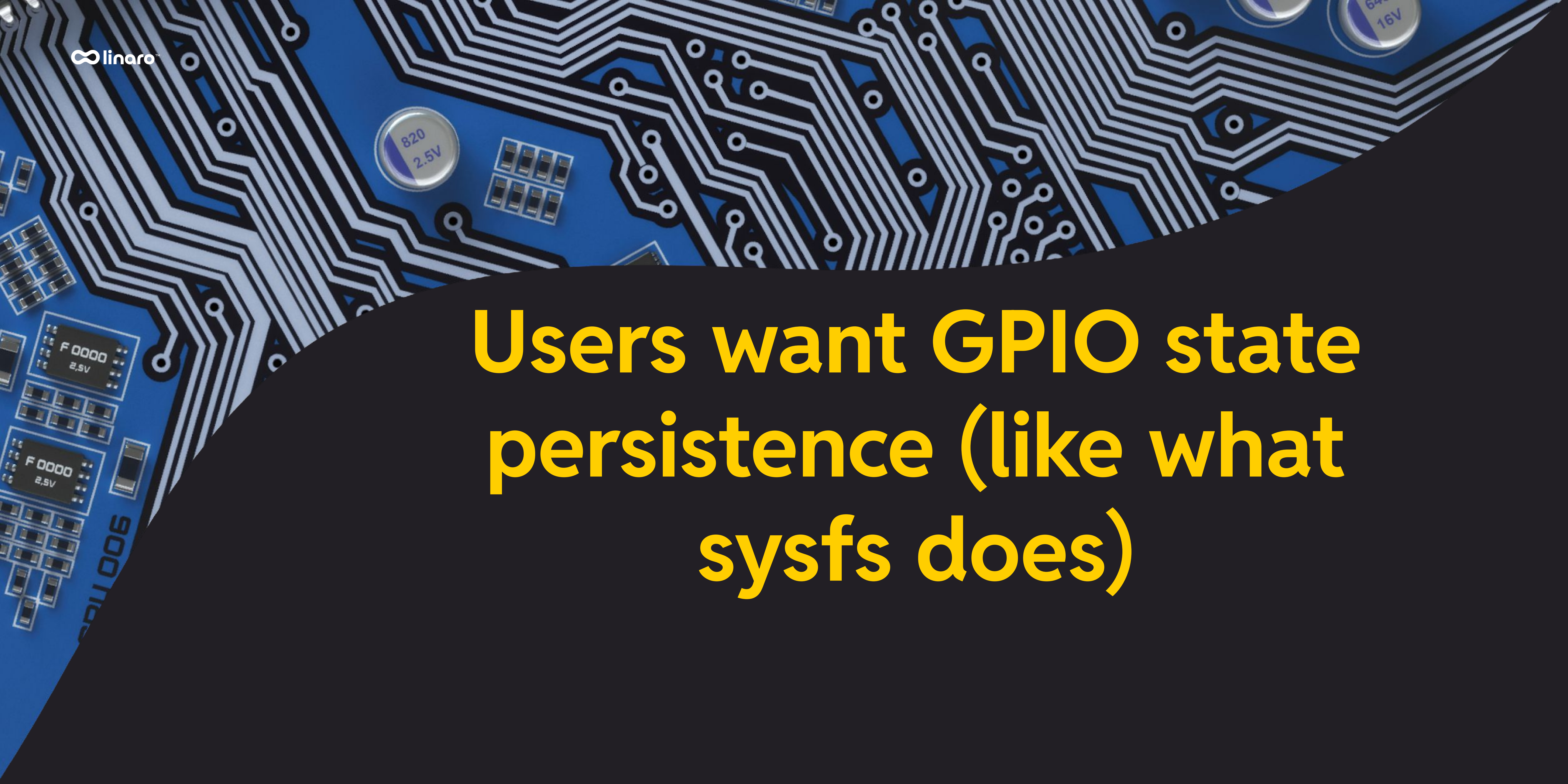
GPIO character device



Users want simplicity

libgpiod & gpio-tools

Users when they realize gpioset does not guarantee persistence



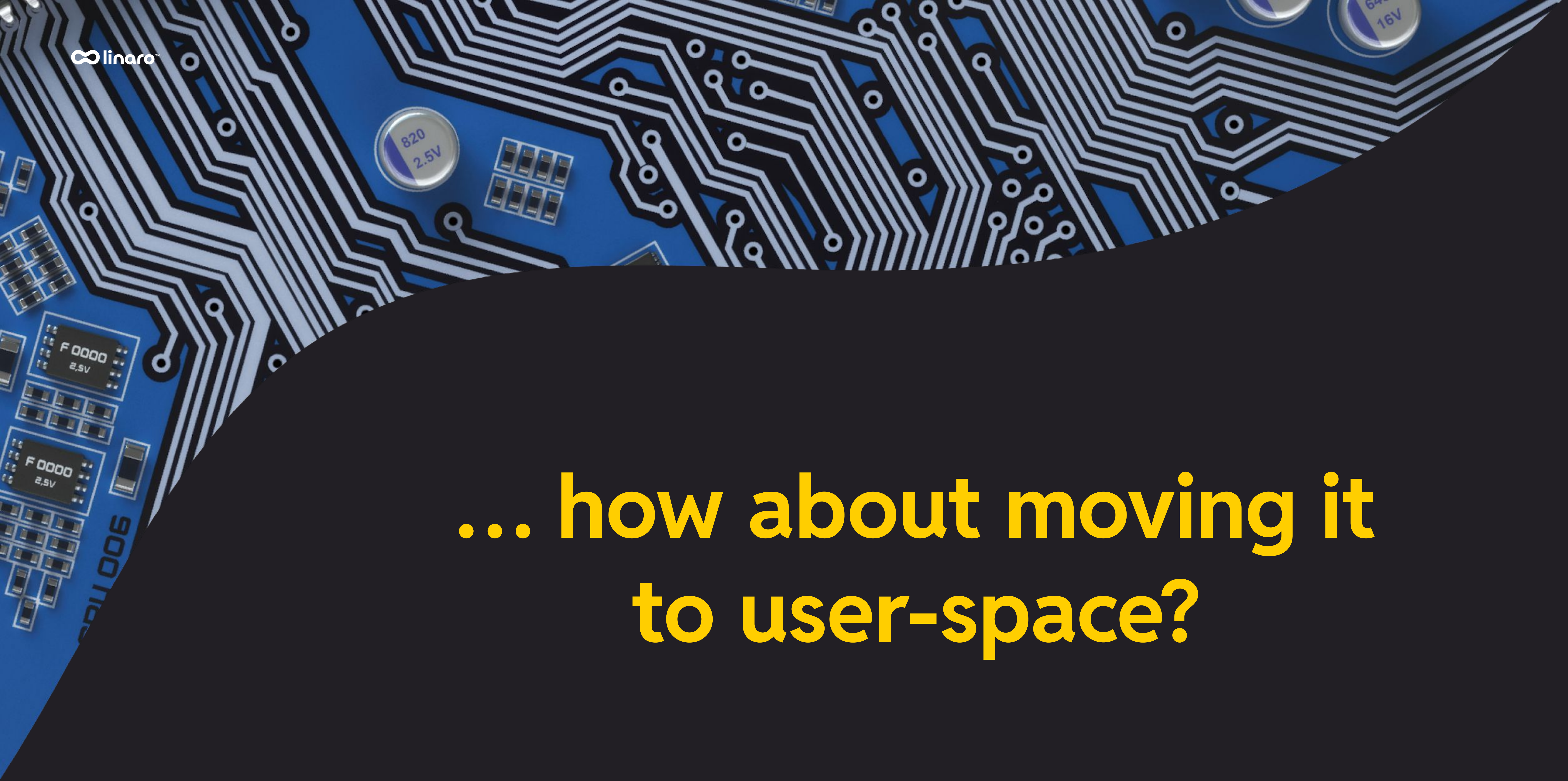
Users want GPIO state persistence (like what sysfs does)

gpio-manager

- Relevant talk:
  - *“Give Me Back My GPIO Persistence - introducing the libgpiod gpio-manager “*
  - https://www.youtube.com/watch?v=tUFcWVwyzQg
- gpio-manager and gpiocli are seeing some adoption
- Users can now do:
  - `gpiocli request –output foobar`
  - `gpioset foobar=active`
  - `gpioget foobar`

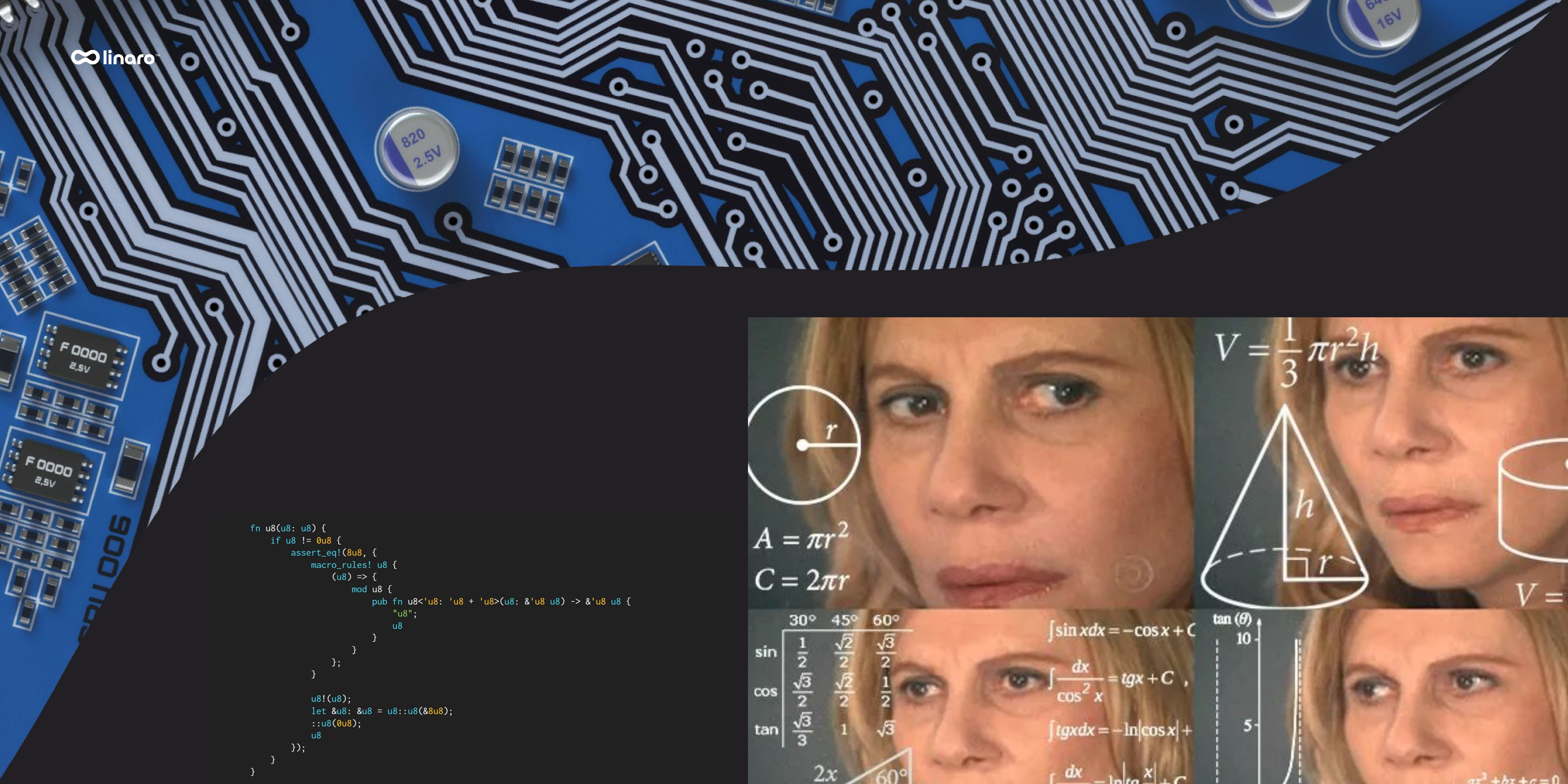
Turns out users just don’t want to change their programs

If you still want to use `/sys/class/gpio`...



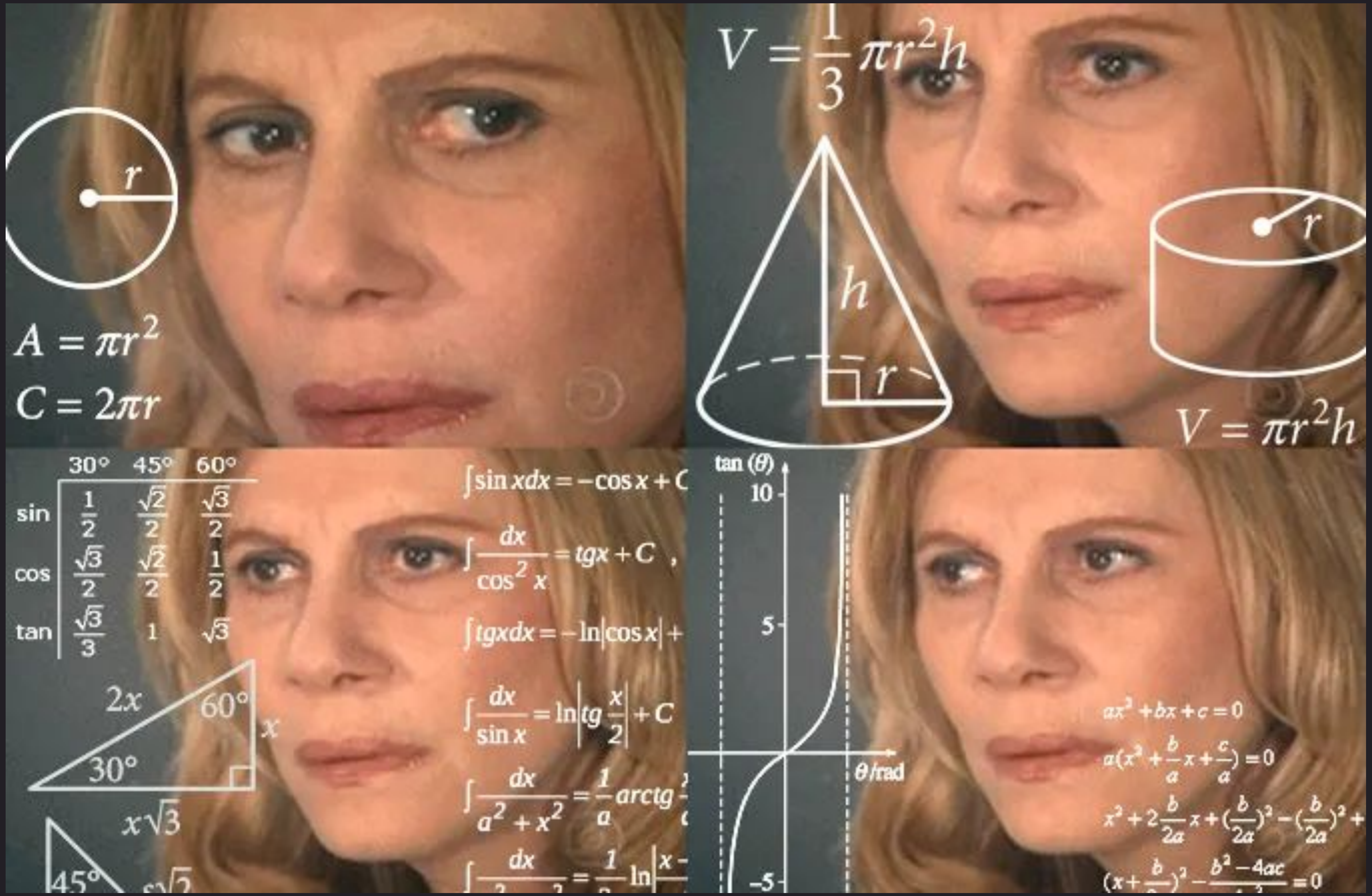
… how about moving it to user-space?

/me should really start learning rust...



```rust
fn u8(u8: u8) {
    if u8 != 0u8 {
        assert_eq!(8u8, {
            macro_rules! u8 {
                (u8) => {
                    mod u8 {
                        pub fn u8<'u8: 'u8 + 'u8>(u8: &'u8 u8) -> &'u8 u8 {
                            "u8";
                            u8
                        }
                    }
                };
            }

            u8!(u8);
            let &u8: &u8 = u8::u8(&8u8);
            ::u8(0u8);
            u8
        });
    }
}
```



Python is good enough! ¯_(ツ)_/¯

Introducing `gpiod-sysfs-proxy`

- user-space compatibility layer for `/sys/class/gpio`
  - uses FUSE to create a filesystem compatible with `/sys/class/gpio` in user-space
  - uses libgpiod to control GPIOs via the character device

Introducing `gpiod-sysfs-proxy`

- user-space compatibility layer for `/sys/class/gpio`
  - uses FUSE to create a filesystem compatible with `/sys/class/gpio` in user-space
  - uses libgpiod to control GPIOs via the character device
- Get it at:
  - https://github.com/brgl/gpiod-sysfs-proxy
  - https://pypi.org/project/gpiod-sysfs-proxy/

Introducing `gpiod-sysfs-proxy`

- user-space compatibility layer for `/sys/class/gpio`
  - uses FUSE to create a filesystem compatible with `/sys/class/gpio` in user-space
  - uses libgpiod to control GPIOs via the character device
- Get it at:
  - https://github.com/brgl/gpiod-sysfs-proxy
  - https://pypi.org/project/gpiod-sysfs-proxy/
- Caveats:
  - Polling the `value` attribute works a bit differently due to libfuse limitations
  - No static GPIO base yet (working on it!)

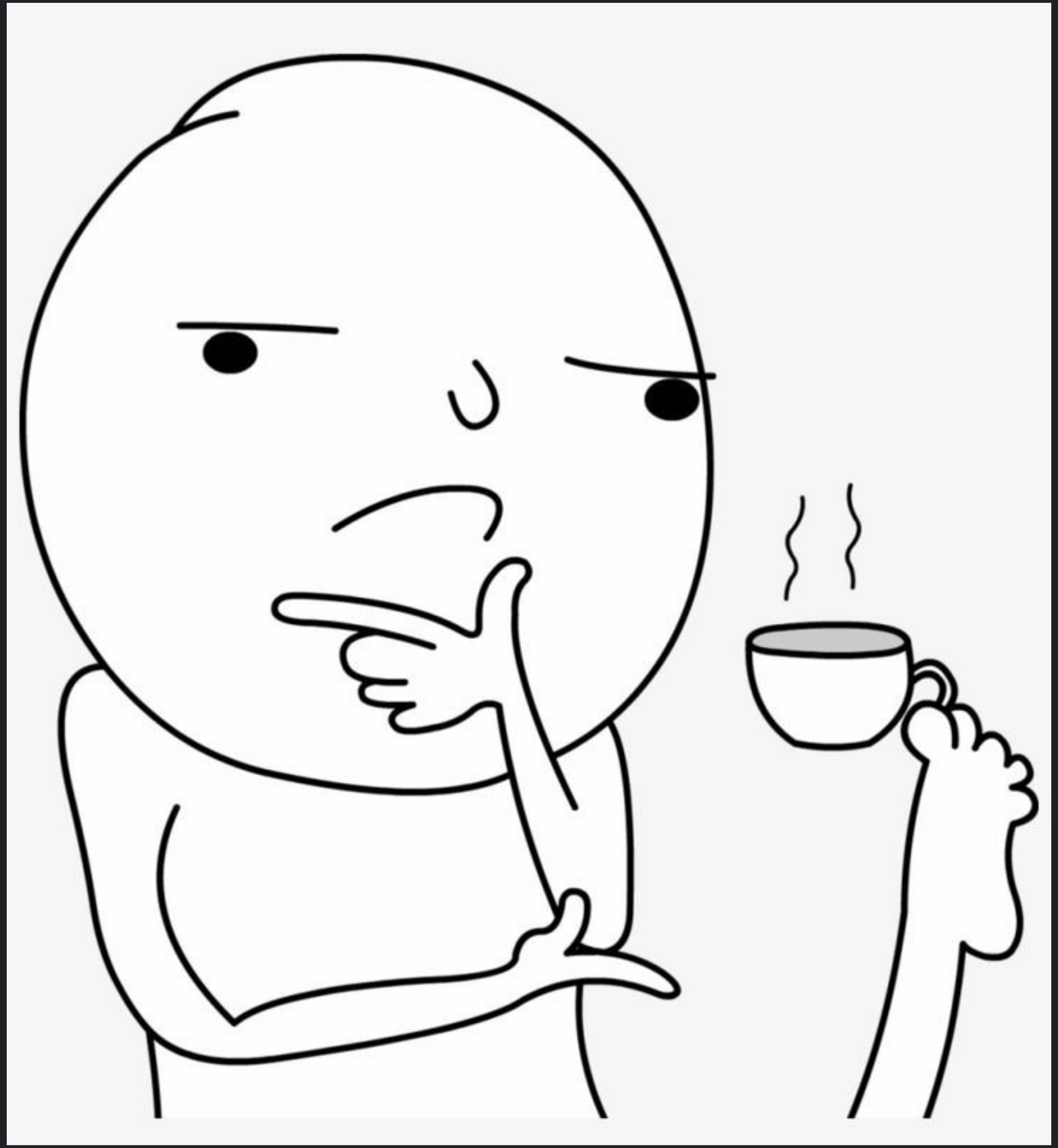
Introducing `gpiod-sysfs-proxy`

- user-space compatibility layer for `/sys/class/gpio`
  - uses FUSE to create a filesystem compatible with `/sys/class/gpio` in user-space
  - uses libgpiod to control GPIOs via the character device
- Get it at:
  - https://github.com/brgl/gpiod-sysfs-proxy
  - https://pypi.org/project/gpiod-sysfs-proxy/
- Caveats:
  - Polling the `value` attribute works a bit differently due to libfuse limitations
  - No static GPIO base yet (working on it!)
- Passes compatibility tests:
  - https://github.com/brgl/gpio-sysfs-compat-tests

`gpiod-sysfs-proxy` usage

- Install using pip3: `pip3 install gpiod-sysfs-proxy`
- Mount at the directory of choice: `gpiod-sysfs-proxy /sys/class/gpio`

But I don't have `/sys/class/gpio`, it's disabled in Kconfig

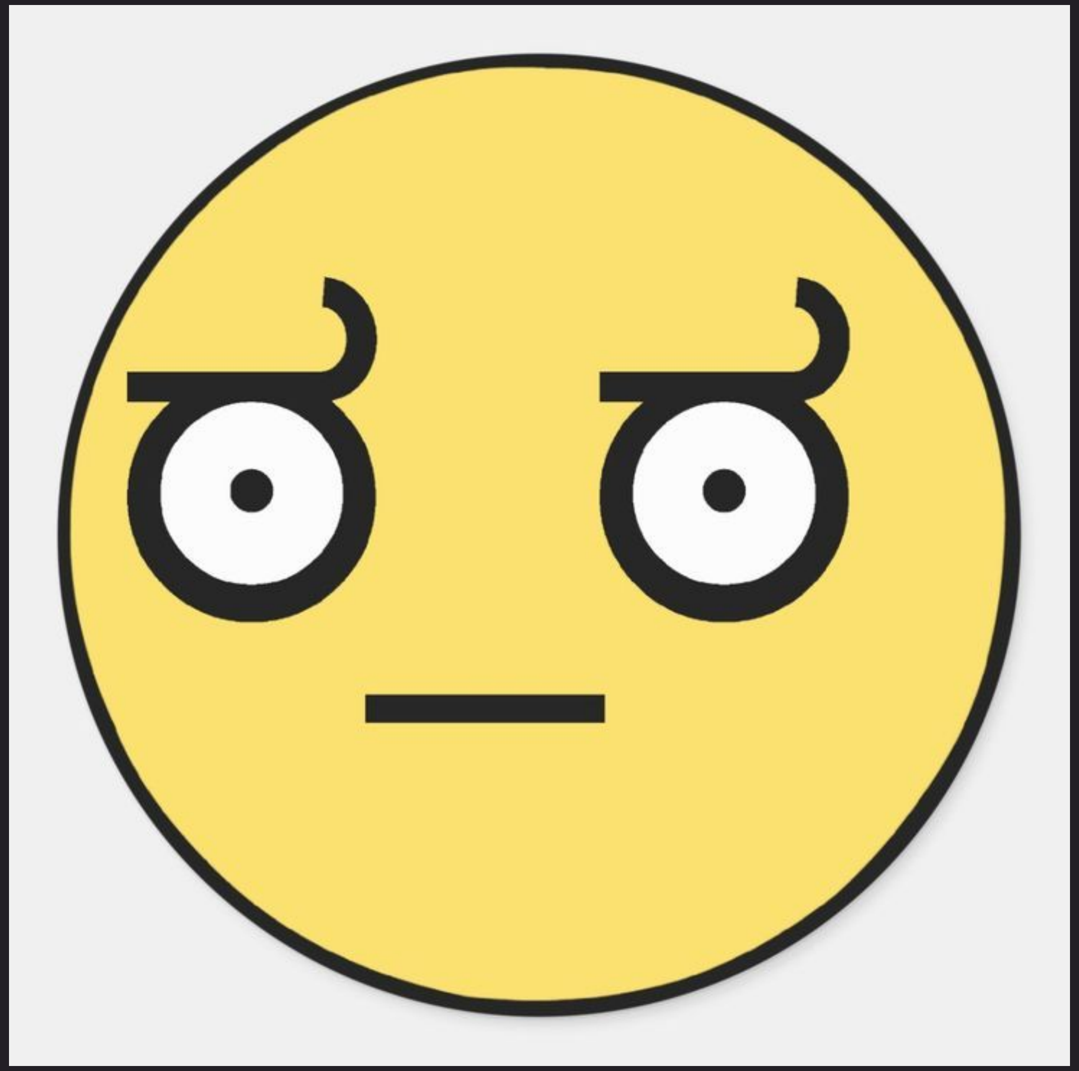


gpiod-sysfs-proxy integration

```
mkdir -p /run/gpio/sys /run/gpio/class/gpio /run/gpio/work
mount -t sysfs sysfs /run/gpio/sys -o nosuid,nodev,noexec
mount -t overlay overlay /sys/class \
    -o upperdir=/run/gpio/class,lowerdir=/run/gpio/sys/class,workdir=/run/gpio/work,nosuid,nodev,noexec,relatime,ro
```

`gpiod-sysfs-proxy` integration

```
mkdir -p /run/gpio/sys /run/gpio/class/gpio /run/gpio/work
mount -t sysfs sysfs /run/gpio/sys -o nosuid,nodev,noexec
mount -t overlay overlay /sys/class \
    -o upperdir=/run/gpio/class,lowerdir=/run/gpio/sys/class,workdir=/run/gpio/work,nosuid,nodev,noexec,relatime,ro
```



What's next?

- Support static GPIO base numbers in gpiod-sysfs-proxy
- Try to get some traction for it
- Still want to learn that rust...
  - Filesystem based GPIO interface that improves upon the sysfs idea?
    - Would have fine-grained permission control that with D-Bus requires a lot of polkit integration and/or using `gpio-aggregator`

Summary

- `/sys/class/gpio` will still be there for a while
  - cannot remove it as long as it has users
- libgpiod offers a bunch of alternatives
- gpiod-sysfs-proxy offers compatibility with `/sys/class/gpio` implemented in user-space with libgpiod
- Converting all kernel drivers to new API will make a stronger case for removal of sysfs

Q & A

Thank You!

Visit linaro.org

Contact me at:

bartosz.golaszewski@linaro.org