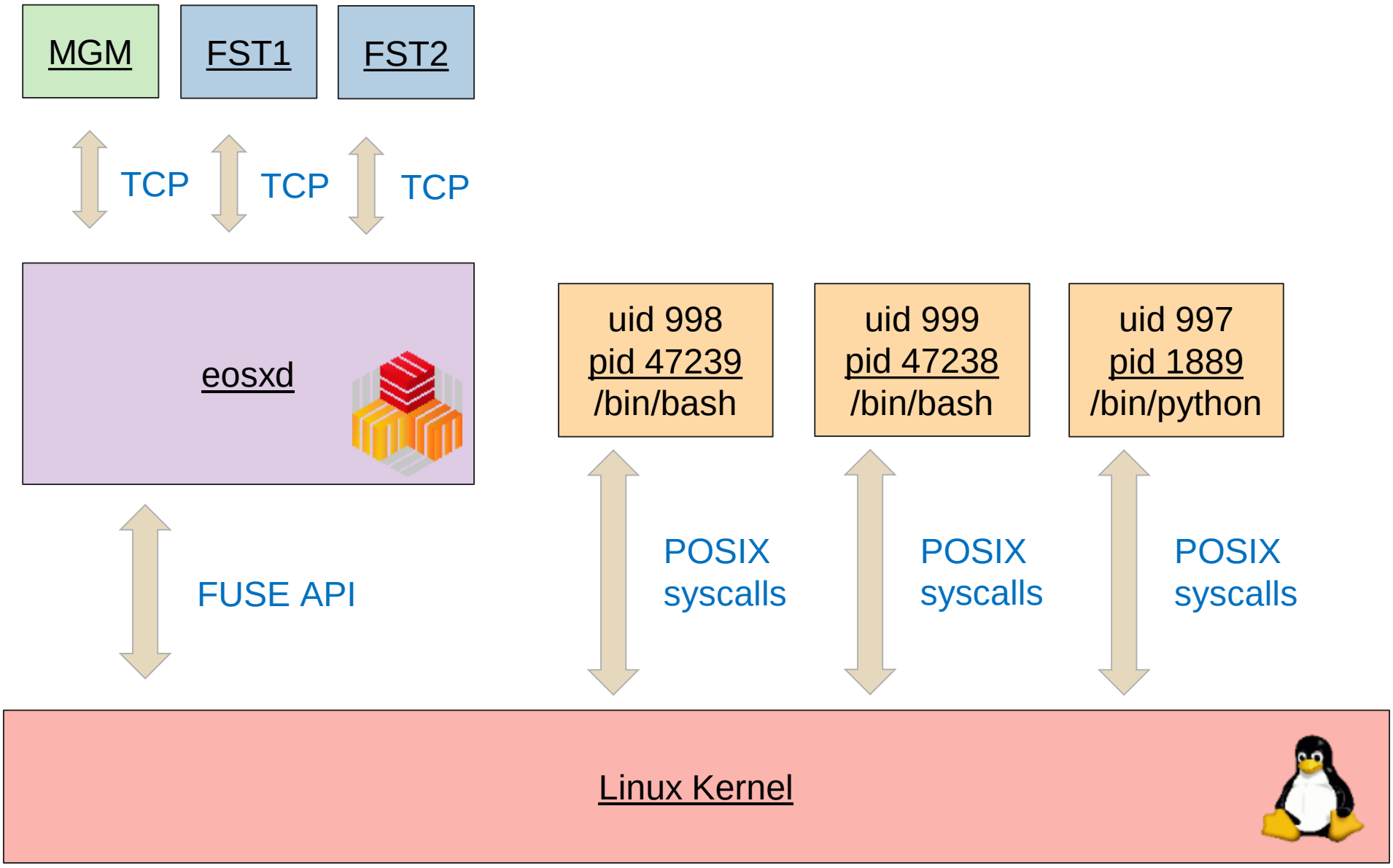
User authentication in eosxd

A tale of `/proc/pid/environ` and kernel deadlocks

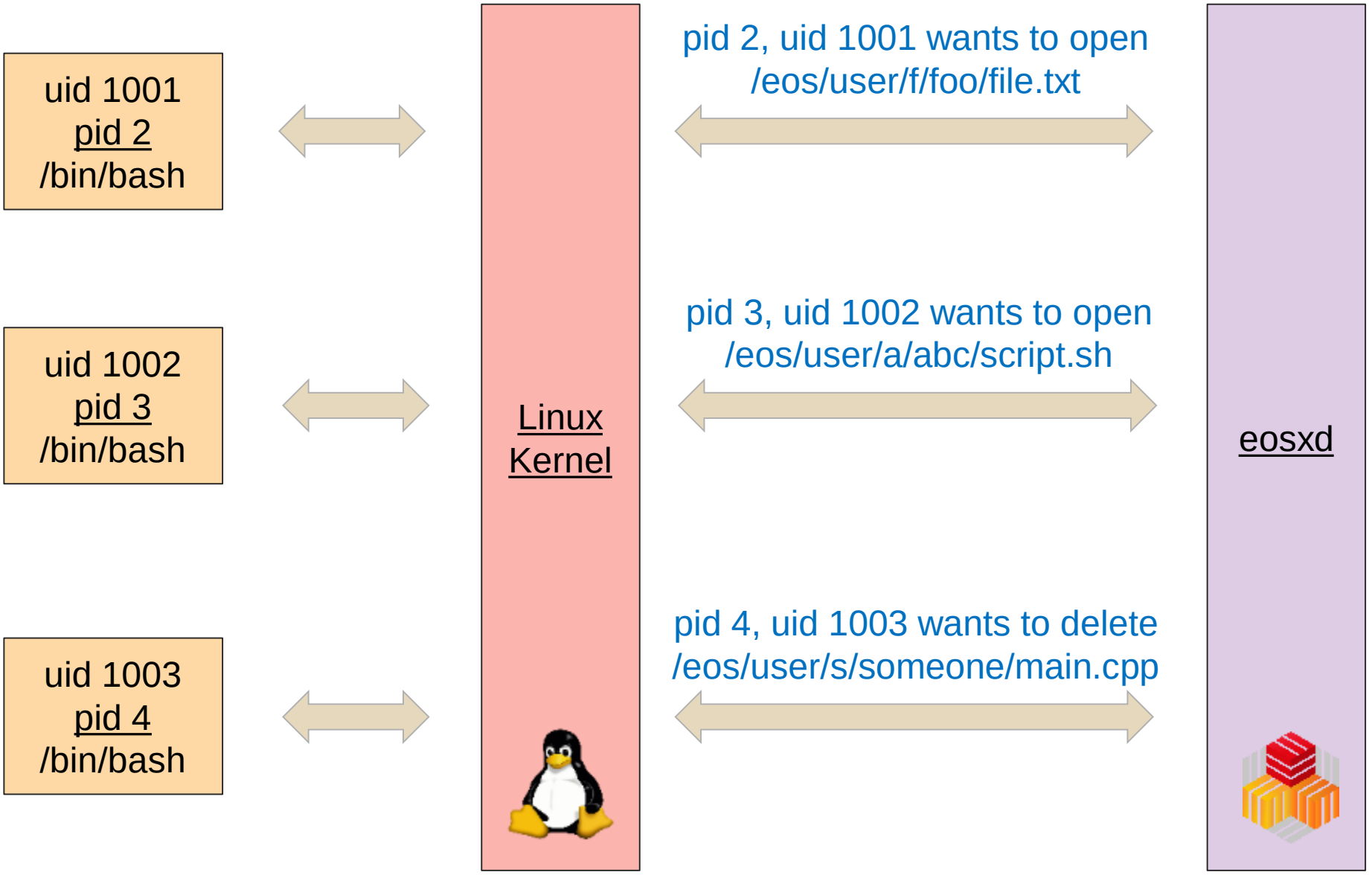
Georgios Bitzes

on behalf of the **EOS team**

Super simplified overview



The core issue



The core issue (2)

- MGM expects a Kerberos ticket, not uid…

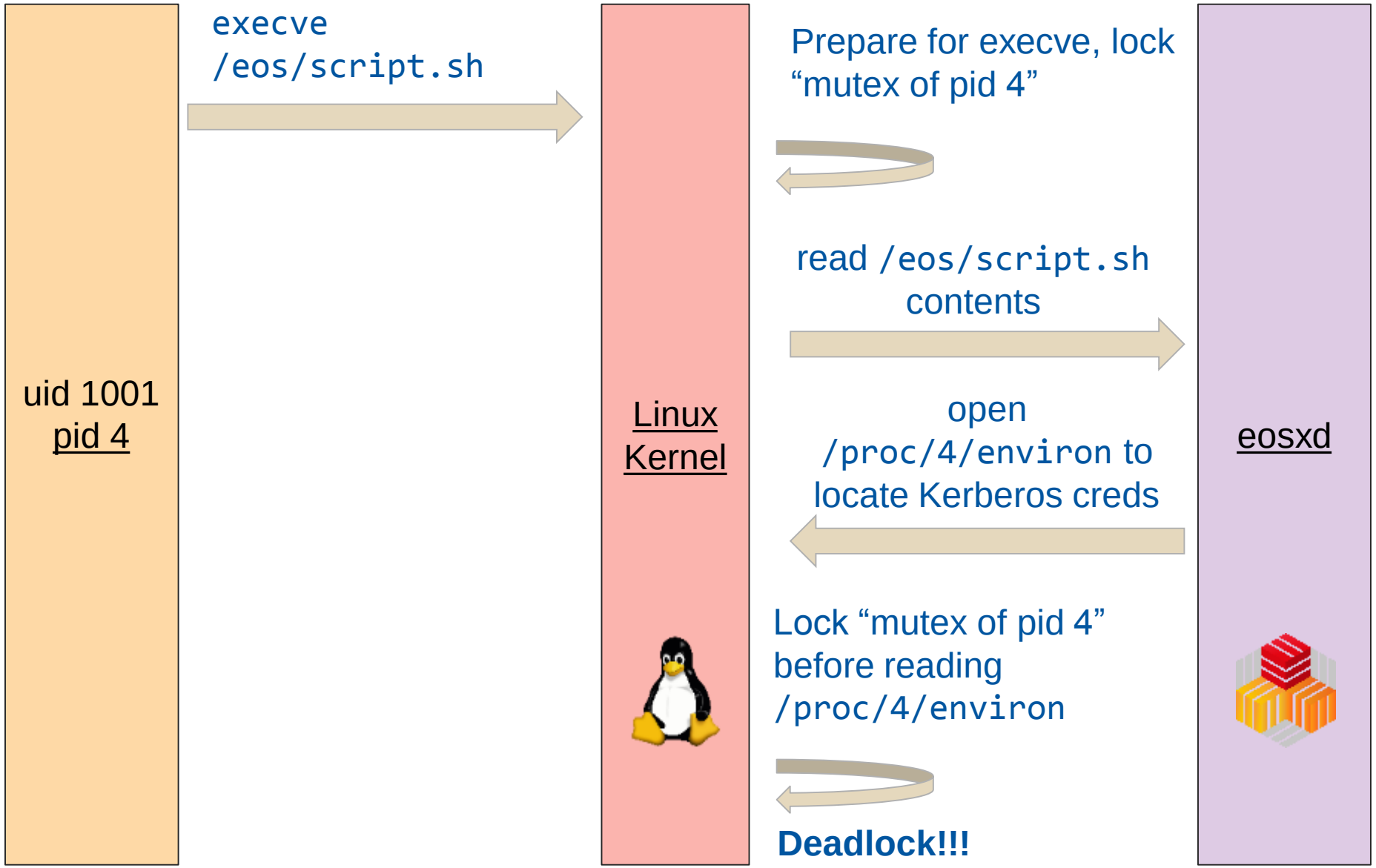
- How to translate (uid, pid) pairs to Kerberos tickets?
  - … preferably without opening horrific security holes in the process :-)

Environment variables

- eosxd runs as **root**… we can read `/proc/pid/environ` of contacting pid, and follow `$KRB5CCNAME`

- Just pay attention not to confuse the permissions…
  - Check that uids of requests and file credentials match, watch out for race conditions when reading files…

What could possibly go wrong?



How to tell if the kernel is inside the execve codepath?

- FUSE API does not contain field “is in execve” …
- If we knew whether reading `/proc/pid/environ` is dangerous, we could work around it
  - Read the environ of parent process instead, process group leader, session leader…

Guessing if caller is inside execve

Guessing if caller is inside execve (2)

- `FMODE_EXEC`: Special hint to the filesystem by the kernel during `open()` that we're reading a file to be used as image for `execve`

- No false positives, but false negatives

- Path lookups sometimes precede `open()`, which don't have this flag… and still cause deadlocks

Guessing if caller is inside execve (3)

- The good news: `FMODE_EXEC` already eliminates the vast majority of deadlocks
- Checking `PF_FORKNOEXEC` flag seems like a good idea too, but may have false positives…
- The two heuristics above reliably prevent **almost all** deadlocks.

The final hack: Reading with a timeout

- Read `/proc/pid/environ` in a separate thread asynchronously, give back `std::future`.
- If it takes more than 100ms, call it a deadlock and try parent's creds, process group leader's, session leader's, etc.
- Once `execve` completes, the thread reading `/proc/pid/environ` unblocks, the read goes through.
  - Deadlock is resolved 

It works

- The deadlocks are a non-issue for end-users.
  - Except for the very rare 100ms penalty when hitting "uncached kernel path lookup during `execve` for a process not in `PF_FORKNOEXEC`" …
  - But hey, this is a network filesystem... nobody will notice… shhhh…

- Alternative: something similar to `eosfusebind`, which adds considerable user-facing complexity.

eosfusebind

- Being deprecated, will not be used in new FUSE daemon
- `/var/run/eosd/credentials` goes away as well

- Kernel deadlocks the original "**raison d'être**" of `eosfusebind`

Is there a perfect solution?

- Yes… check in_execve bit

```
struct task_struct {
  …
  /* bit to tell LSMs we're in execve */
  unsigned in_execve:1;
  …
}
```

- Problem: It’s in kernel space…

Is there a perfect solution? (2)

- Load kernel module to help read in_execve
  - expose /proc/pid/in_execve or something
  - Problem: in_execve is unsynchronized…
    - only the calling syscall codepath can access it safely without data races, even at the kernel level
  - Would need changes to core kernel structure, `task_struct`…
    - I am not brave enough to submit such a patch
    - Even then, would be many years before change lands in our platforms

Thanks

To admire the aforementioned hacks and workarounds in all their glory, visit:

- https://gitlab.cern.ch/dss/eos/tree/master/fusex/auth

Questions, comments?

Georgios.bitzes@cern.ch


CERN