

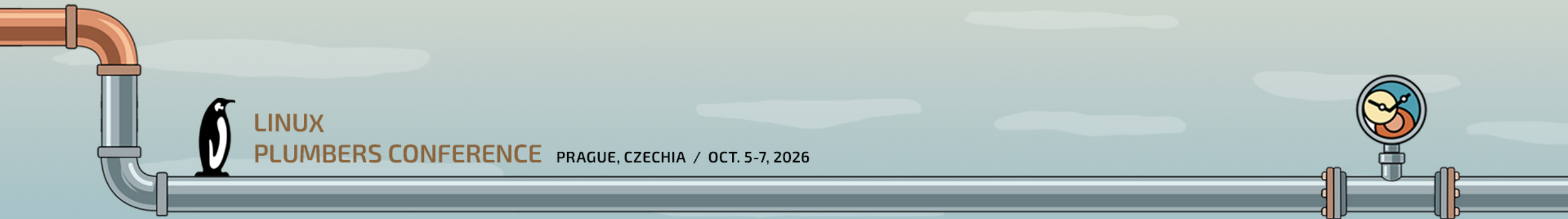
How to support other memory providers for `guest_memfd`

Contact ackerleytng@google.com if you have questions/suggestions!



2 components

1. Blocks host mappings of CoCo guest-private memory
2. Actually provides the memory



Discussion overview

- Scope: What kinds of providers should `guest_memfd` support?
- Userspace interface
- In-kernel interface. Want to take away a list of
 - Functions provider should expose to `guest_memfd`
 - Functions `guest_memfd` should expose to provider



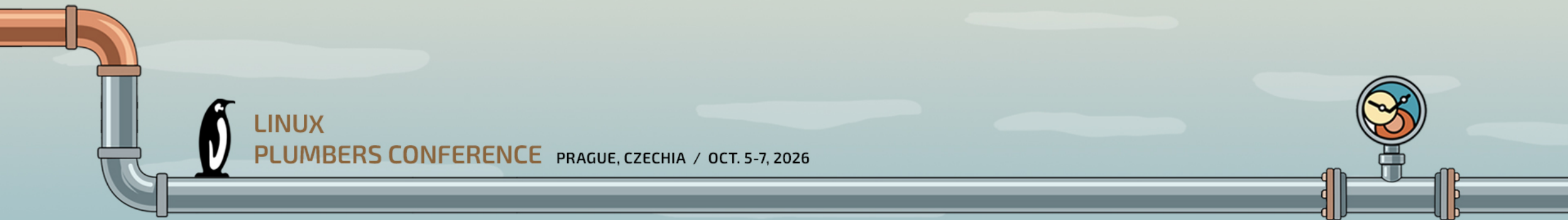
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



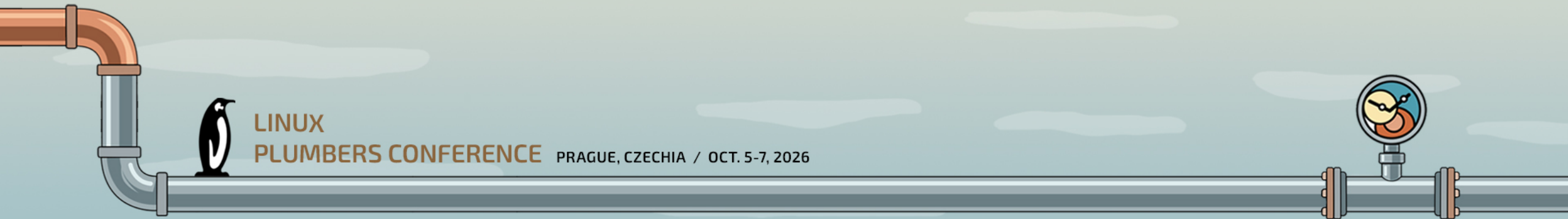
Scope: Classes of resources

- Host memory: e.g. HugeTLBfs
- PFNs: e.g. DAX (David Woodhouse, Frank, Dave Jiang)
- Host MMIO
- Did I miss any other classes of resources?



Constraints

- No "backdoor" host access to guest-private memory



Proposal: Create `guest_memfd` with resource fd

- HugeTLBfs/tmpfs: Resource *pool* handle:
 - `fsopen()` → `fsconfig(mpol)`, `fsconfig(page_size)` → `fsmount()`
 - Pass `mount_fd` to create `guest_memfd`.
- In kernel: look up some `provider_ops`, allocate resource



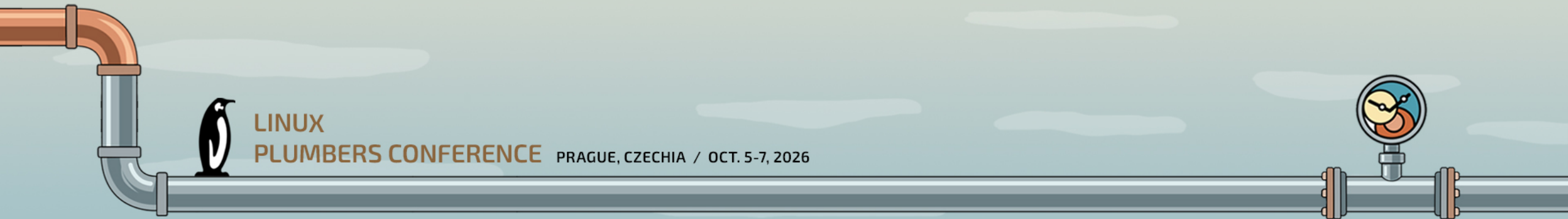
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



In-kernel API: Requirements

- Ideally: `guest_memfd` needs no changes to support a new provider
- Should support providers that are loaded after boot as modules (Frank)
- Cover most filesystems, add ops and parameters when required
- Any more that people might want to call out?



In-kernel API: Functions provider should expose

```
struct provider_operations {  
    .allocate  
    .free  
    .attach  
    .release  
};
```



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Roles: `.allocate()` + `.free()`

<code>guest_memfd</code>	provider
Provide <code>mpol</code>	Allocate <code>folio</code> according to <code>mpol</code>



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Roles: `.allocate()` + `.free()`

<code>guest_memfd</code>	<code>provider</code>
Provide <code>mpol</code>	Allocate <code>folio</code> according to <code>mpol</code>
Handle <code>inode</code> usage/stats	Handle provider usage tracking/stats



Roles: Handling events: Memory failure

- `memory_failure()` uses `folio->mapping->a_ops` to determine how to handle memory failure.
- `guest_memfd` should tell KVM to zap stage 2 page tables
 - Then notify the provider



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Roles: Handling events: Reclaim

- Currently, `guest_memfd` doesn't participate in LRU
- Option: continue to not support reclaim
 - Example: tmpfs does. Should we require `noswap` ?
- Option: support reclaim
 - Forward reclaim callback to provider?



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



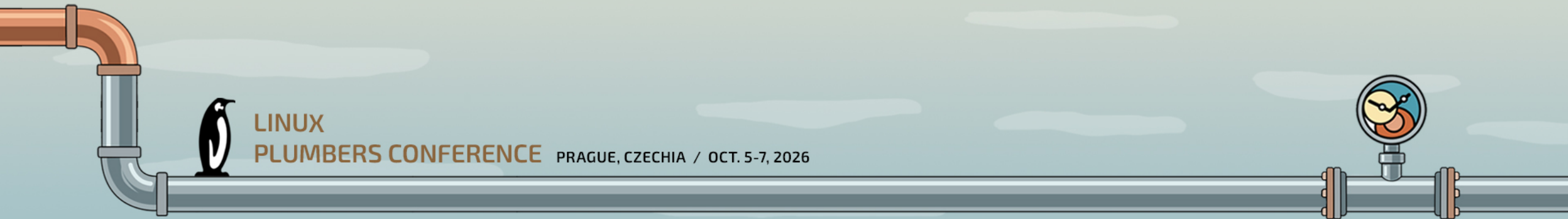
In-kernel API: Fns exposed by `guest_memfd`

- Need to allow provider to revoke memory (David Woodhouse)
- Requirement for DAX memory wrapping device memory too
- Proposal: notifier system
 - Provider notes down a `guest_memfd` pointer in `.attach()`
 - Provider notifies `guest_memfd` that it is revoking memory
 - `kvm_gmem_provider_revoke(inode, offset, size)`



In-kernel API: What else will the provider need?

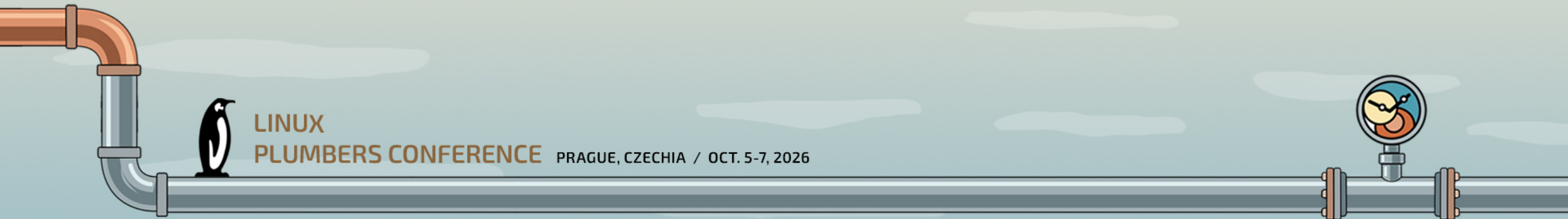
- What else will the provider need `guest_memfd` to do?



Thank you for your time!

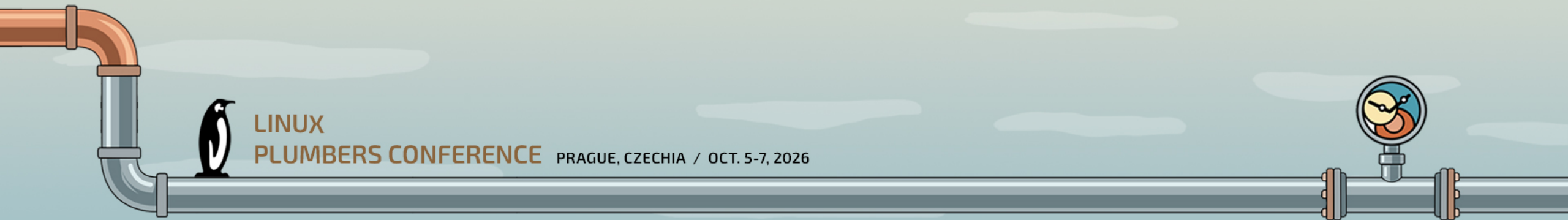
Contact ackerleytng@google.com if you have questions/suggestions!

- Ackerley's RFC: <https://lore.kernel.org/all/20260925-gmem-tmpfs-backend-v1-0-d36159822d18@google.com/>
- David Woodhouse (`guest_memfd` + PFN allocator): <https://lore.kernel.org/all/2026072011259.122911-1-dwmw2@infradead.org/>
- Dave Jiang (`guest_memfd` + DAX): <https://lore.kernel.org/all/20260423170219.281618-1-dave.jiang@intel.com/>
- Frank van der Linden (`guest_memfd` + `ZONE_DEVICE`): <https://lore.kernel.org/all/20260424172912.548636-1-fvdl@google.com/>



Confidential IO

- Needs MMIO address passthrough
- Now we have private MMIO, host should not write private MMIO addresses
- Different from private system memory: no runtime conversion



Host MMIO addresses as a resource

- Current proposal: have different dmabuf fds for shared/private ranges
 - KVM must now accept dmabuf fds describing a private memslot
 - Or tell KVM somehow that this memslot is private-only
- `guest_memfd` + resource fd
 - 1 `guest_memfd` for entire device
 - KVM continues to fault stage 2 via `guest_memfd`
 - `.can_convert()` callback to block conversions?



Looking up ops: `struct super_operations`

- Could put the ops in `struct super_operations`

```
static const struct super_operations foobar_super_ops = { ...  
    .gmem_provider_ops = &foobar_gmem_provider_ops;  
}
```

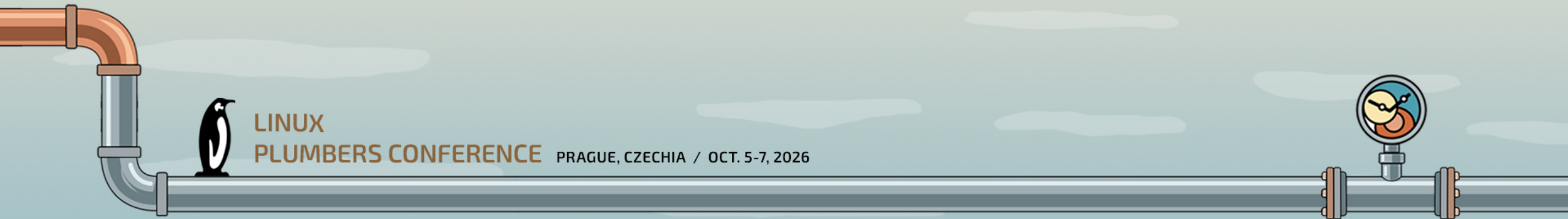
- Or for wider usage (not KVM/ `guest_memfd` focused)

```
static const struct super_operations foobar_super_ops = { ...  
    .attach = foobar_gmem_attach;  
    .release = foobar_gmem_release;  
}
```



Looking up ops: registry

- Have table of `{ FOOBAR_MAGIC: foobar_ops }` in the kernel
 - Register filesystems as they are loaded (as modules)
- Could also have registry in KVM, but KVM is a module too...



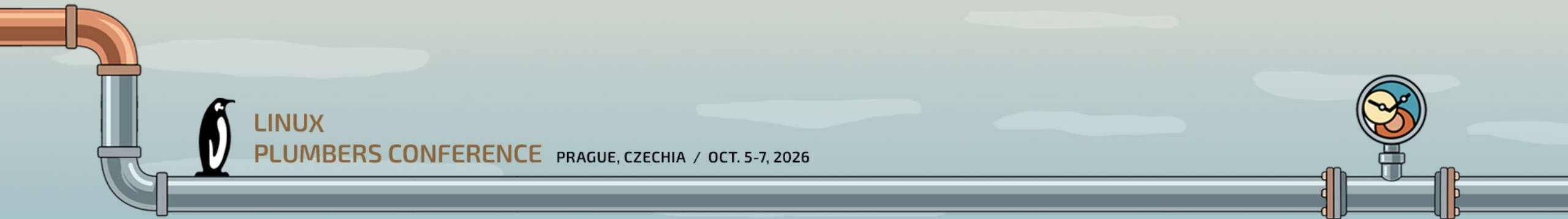
Userspace API 1: More flags (Sean said no)

- `GUEST_MEMFD_FLAG_HUGETLB` , `GUEST_MEMFD_FLAG_DAX` , etc
- Even MORE config knobs:
 - HugeTLB: page size, which mount?

```
args = {  
    .flags      = GUEST_MEMFD_FLAG_DAX,  
    .dax_region = 0,  
    .dax_id     = 0,  
    .dax_align  = SZ_2M  
    .dax_size   = 1ULL << 30,  
};  
  
ioctl(kvm_fd, KVM_CREATE_GUEST_MEMFD, &args);
```

Userspace API 1: More flags: Cons

- Too many knobs: reinventing the interfaces other filesystems have
- Hard to generalize, complex to support filesystems loaded as modules



Userspace API 2: `backing_fd`

```
/* Set up mount however you want. */

backing_fd = open("/mnt/huge1g/some_file", O_RDWR | O_CREAT, 0600);
ftruncate(backing_fd, size_fixed_before_creating_gmem);

args = {
    .backing_fd = backing_fd,
};

ioctl(kvm_fd, KVM_CREATE_GUEST_MEMFD, &args);
```



Userspace API 2: `backing_fd`: Backdoors!

- Backdoors:
 - `write(backing_fd)` itself
 - Opening the same file again
- Seal `backing_fd` to block all future `mmap` / `read` / `write` / `fallocate`



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Userspace API 3: udmabuf-style pinning

- Provide source file, specify range to use with `guest_memfd`
- `guest_memfd` pins the folios that were specified
- Cons:
 - Still need to solve backdoor access issue
 - Can't support runtime faults (must preallocate)



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Userspace API 4: `copy_file_range`

- Transfer/donate the folios from source fd to `guest_memfd`
- `copy_file_range` (existing syscall) is an in-kernel copy between fds
- New flag `MOVE` : completely truncate the folios from source fd and insert it into `guest_memfd`
- Pro: Allocations are all handled directly through original fd
- Con: Can't support runtime faults (must preallocate)
- Con: Awkward role split: handling `mpol` , `memcg` charging...



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Introduction: What is `guest_memfd`

- `guest_memfd` is KVM's guest-first memory provider
- Created for CoCo VMs, will support all VMs



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Introduction: Using `guest_memfd`

- KVM `ioctl` : create a `guest_memfd` , specify size.
- Configure `guest_memfd` in a `memslot` to map some GPA range to `guest_memfd` offset.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

