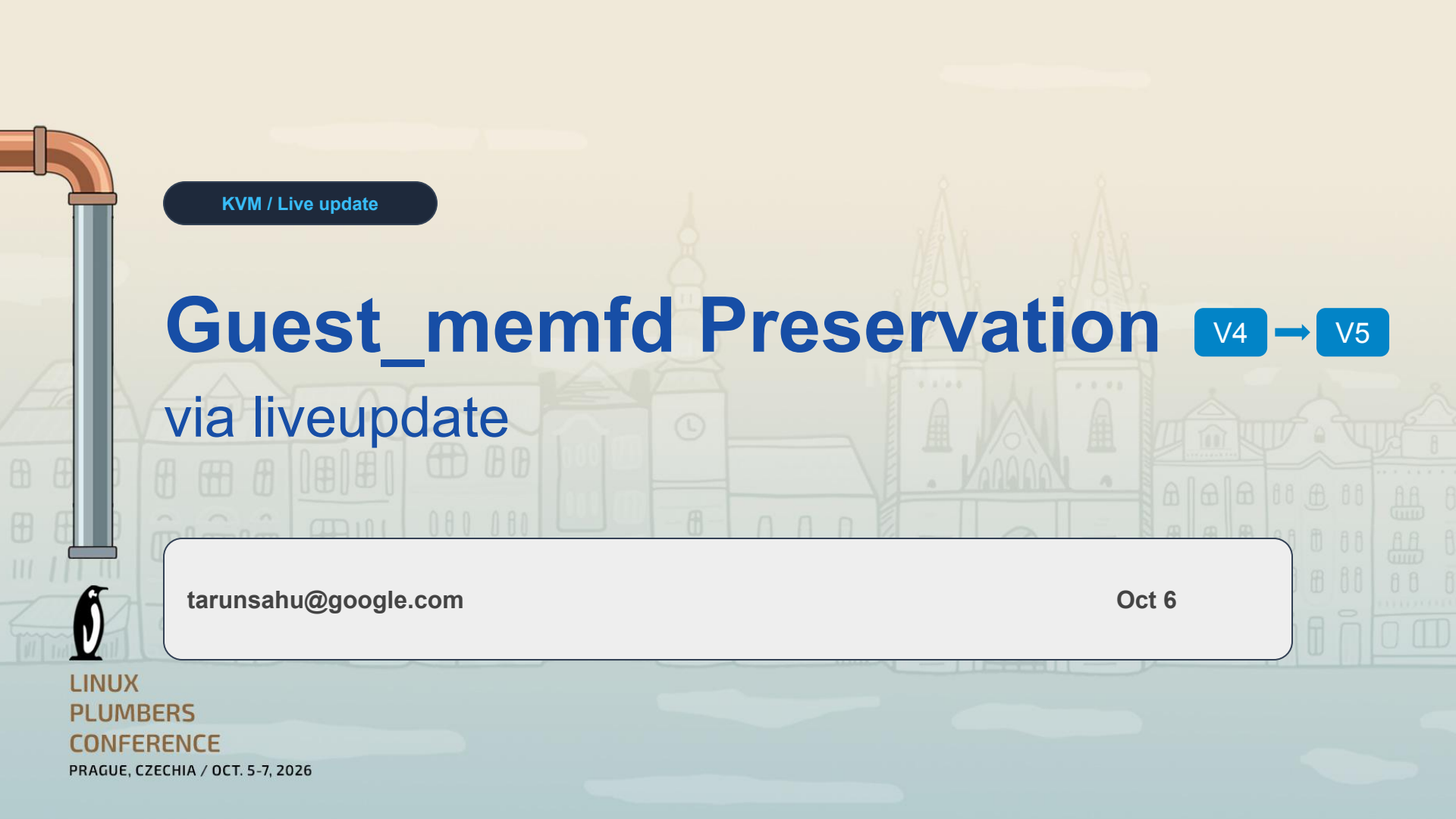




KVM / Live update

Guest_memfd Preservation

V4



V5

via liveupdate

tarunsahu@google.com

Oct 6



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Brief About `guest_memfd`

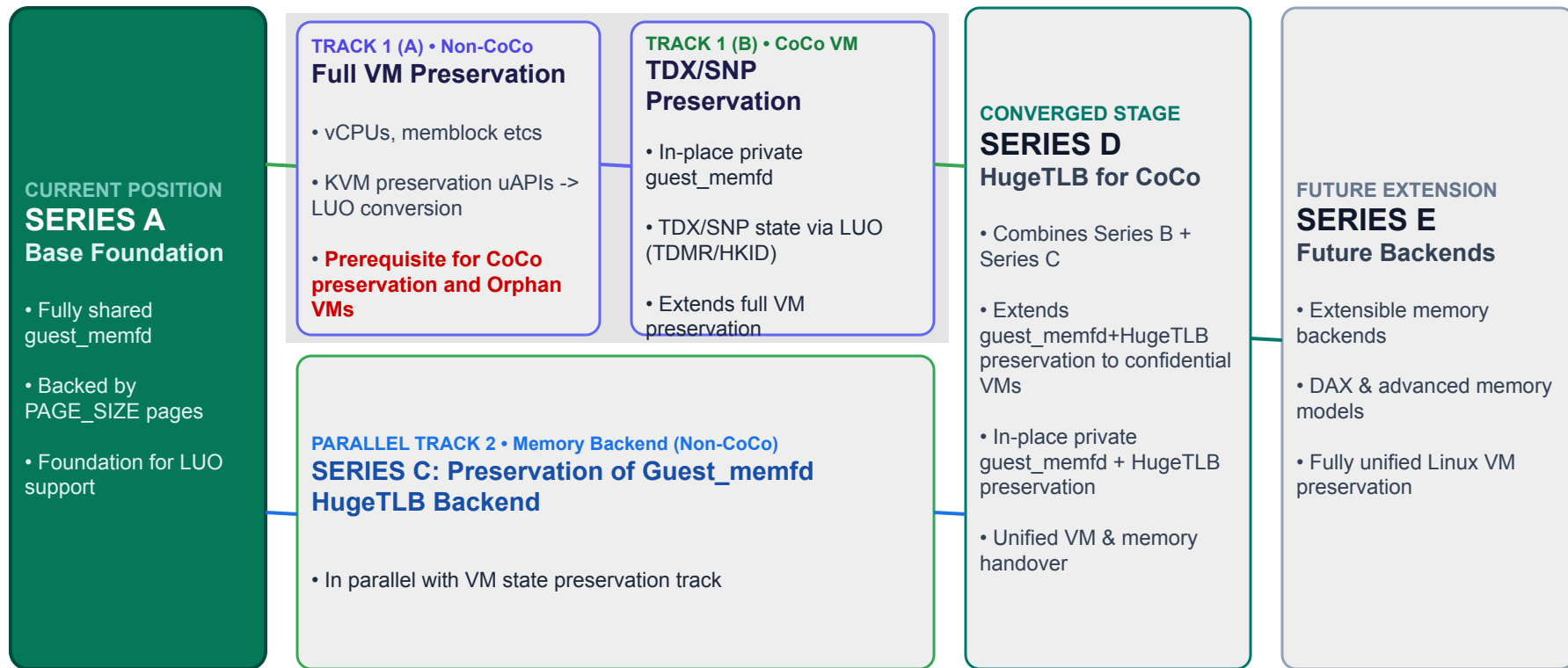
- fd based guest-first memory subsystem in KVM for VMs
- Private to guest and inaccessible to host, unless explicitly shared
- Primarily created for confidential VMs (private), now extended for non-CoCo VMs(shared)
- Created via IOCTL `KVM_CREATE_GUEST_MEMFD`

Brief About `guest_memfd`

- Currently supports either fully private or fully shared memory
- In-place page conversions merged into KVM x86 coco tree
- Multiple backends support to abstract `guest_memfd` for different memory providers
(Shmem, hugetlb, DAX etc)

guest_memfd Preservation: High Level Roadmap

Incremental progression from base shared foundation to parallel tracks, HugeTLB integration, and unified preservation



Base Foundation Series (Current v4: [Here](#))

Goal:

- Preserve fully shared guest_memfd across kexec (non-cocoVM usecase)
- Set the foundation for VM preservation via LUO (will discuss in next slides)

How:

- Simply walk through the folios in guest_memfd inode
- preserve their pfn and index via KHO (ABI details in next slides)

Design Overview

Preservation & Retrieval Workflow Across Kexec

1. PRESERVATION

VM File Preservation

1. `vm_type` **

Freeze `gmem_inode`

- To prevent fallocation and fault (Pages are preserved)
- Any new faults will park the vCPU



Guest_memfd Preservation

- `preserve folios`
- LUO token of associated `vm_file`

↓ Kexec Transition

2. RETRIEVAL

VM File Retrieval

1. Create new vm file

Note: Provides `struct kvm`

Guest_memfd Retrieval

1. Find `vm_file` from session using preserved token
2. Create `guest_memfd` file
3. Populate folios data

** Planning to drop it in next version to leave an empty ABI or no VM preservation at all with LINK IOCTL.

ABI Planning Strategy

To avoid ABI planning for VM in gmem preservation series, which approach should be preferred?

APPROACH 1

Empty VM Preservation

Is it okay?

KEY CONSIDERATION

Maintains current architecture dependency while deferring VM ABI decision from this series.

APPROACH 2

Use KVM GMEM LINK ioctl

ioctl Patch Proposal: Transfers file for live migration

- Retrieve `guest_memfd` with dummy `struct kvm` and link (`ioctl`) to original `kvm`.
- Helps separate `guest_memfd` and VM preservation.

PRO: Removes dependency on VM preservation.

CON: 1. Dummy `vm_file(struct kvm)` required.
2. Link `ioctl` returns different fd. No functional issue.

ABI (PATCH [v4])

VM FILE

```
+struct kvm_luo_ser {  
+    u64 vm_type; **  
+} __packed;
```

** Planning to drop it in next version to leave an empty ABI or no VM preservation at all with LINK IOCTL.

GUEST_MEMFD FILE

```
+struct guest_memfd_luo_folio_ser {  
+    u64 pfn:52;  
+    u64 flags:12;  
+    u64 index;  
+} __packed;
```

```
+struct guest_memfd_luo_ser {  
+    u64 size;  
+    u64 flags;  
+    u64 nr_folios;  
+    u64 vm_token;  
+    struct kho_vmalloc folios;  
+} __packed;  
+
```

Qns

```
+struct guest_memfd_luo_folio_ser {  
+    u64 pfn:52;  
+    u64 flags:12;  
+    u64 index;  
+} __packed;
```

```
+struct guest_memfd_luo_ser {  
+    u64 size;  
+    u64 flags;  
+    u64 nr_folios;  
+    u64 vm_token;  
+    struct kho_vmalloc folios;  
+} __packed;  
+
```



```
+struct guest_memfd_luo_folio_ser {  
+    u64 pfn:52;  
+    u64 flags:12;  
+    u64 index;  
+} __packed;
```

```
+struct guest_memfd_luo_ser {  
+    u64 size_ser;  
+    u64 feature_flags;  
+    u64 size;  
+    u64 flags;  
+    u64 nr_folios;  
+    u64 vm_token;  
+    struct kho_vmalloc folios;  
+} __packed;  
+
```

Qns: mpol should be included in this?

BoF Discussion

Remove Versioning → Use only `guest_memfd` file handler string as sole compatibility measure.

Future Qns for Compatibility: (Independent of Guest_memfd preservation)

How will userspace know which file handler to use for preserving the file, and how will the new kernel know which file handler to use for retrieval?

Proposal:

1. Add new `/dev/liveupdate` ioctl to select an FLB.
2. Extend existing `PRESERVE_FD` ioctl to select File Handler.
3. `PRESERVE_FD` fails if selected File Handler is not registered with selected FLBs

Qns ?

Appendix

Link IOCTL: Pseudo Code [go back](#)

VMM

```
retrieved_fd = luo_retrieve(session, GMEM_TOKEN);  
linked_fd = ioctl(actual_vm_fd, KVM_LINK_GUEST_MEMFD, { .fd =  
retrieved_fd });  
close(retrieved_fd); /* Free dummy KVM immediately */
```

Retrieval
Function

```
+ dummy_vm_file = kvm_create_vm_file(0, "dummy-gmem-vm");  
+ if (IS_ERR(dummy_vm_file)) {  
+     err = PTR_ERR(dummy_vm_file);  
+     goto err_free_ser;  
+ }  
+ dummy_kvm = dummy_vm_file->private_data;  
  
+ file = __kvm_gmem_create_file(dummy_kvm, ser->size, gmem_flags);  
+ /*  
+  * __kvm_gmem_create_file() takes its own kvm_get_kvm() reference.  
+  * Releasing dummy_vm_file leaves dummy_kvm lifetime pinned solely by gmem.  
+  */  
+ fput(dummy_vm_file);
```

NO Preservation & Retrieval Order Required: VM & guest_memfd

1. PRESERVATION ORDER

- `guest_memfd` `luo` needs `VM_TOKEN` of preserved `VM_fd`
- Ans: `guest_memfd_luo_freeze` just before `kexec`

2. RETRIEVAL ORDER

- `guest_memfd` needs `struct kvm`
- Ans: `luo_get_file_incoming` gives `vm_fd` the file from the session