

Lightweight Specification
of Production Systems

Ben Simner Kayvan Memarian

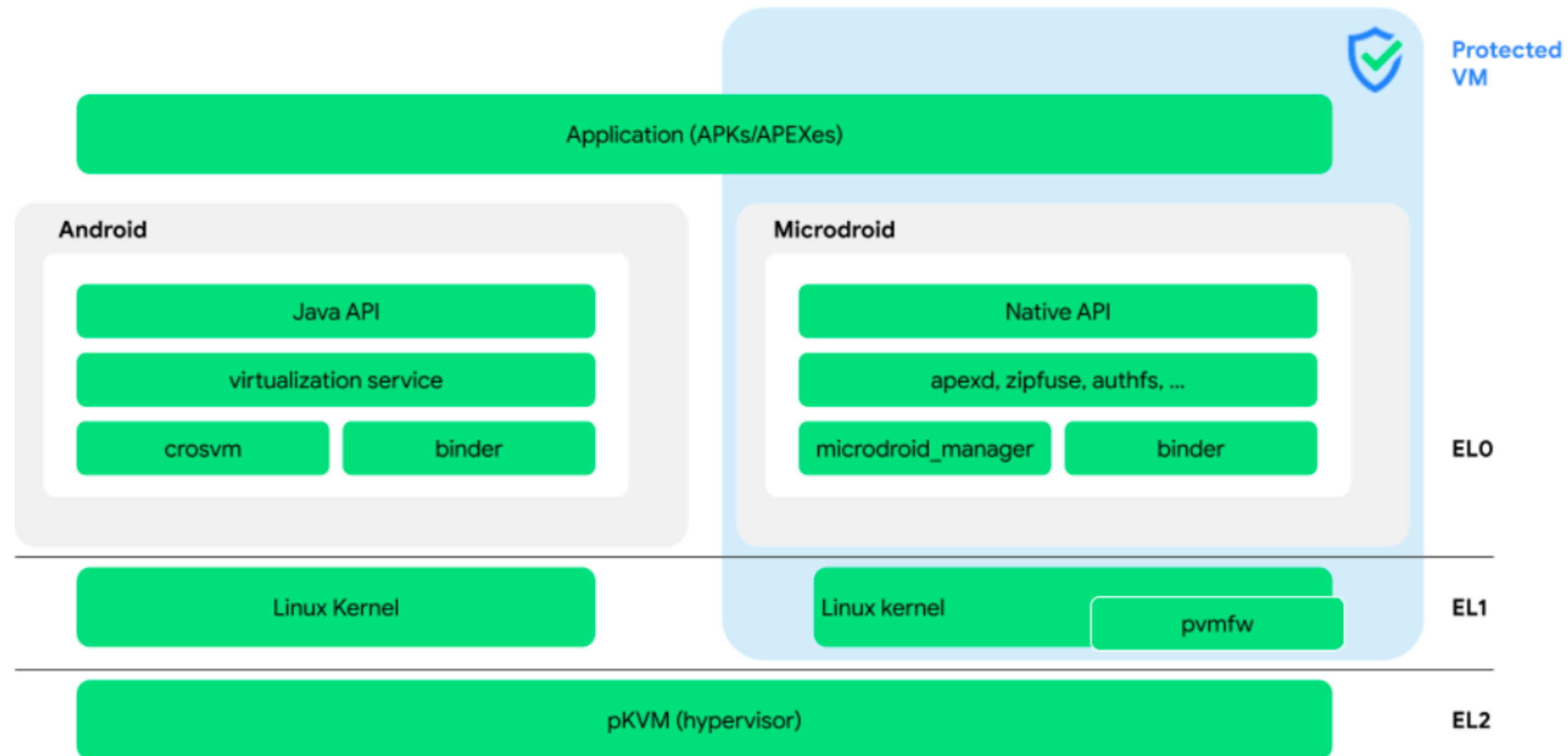
'exec spec'



'caseMate'



Android



[<https://source.android.com/docs/core/virtualization>]

Android



[<https://source.android.com/docs/core/virtualization>]

production systems?

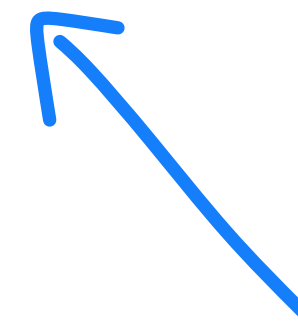
can't prove?

production systems?

can't prove? ... specify anyway!

production systems?

can't prove? ... specify anyway!



use to test

Android 

Software

arm

Architecture

Android 

Software

Top-down: μ KVM API ↓

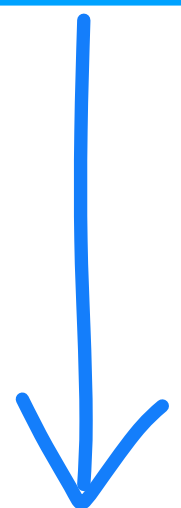
Architecture

arm

Android 

Software

Top-down: pKVM API
Hypercalls, traps



arm

Architecture

Android 

Software

Top-down: pKVM API
Hypercalls, traps

Bottom-up: H/W & Config
Manage

Architecture

arm

Android 

Software

Top-down: pKVM API
Hypercalls, traps

Bottom-up: H/W Config & Manage
Pgtables, TLB invalidation

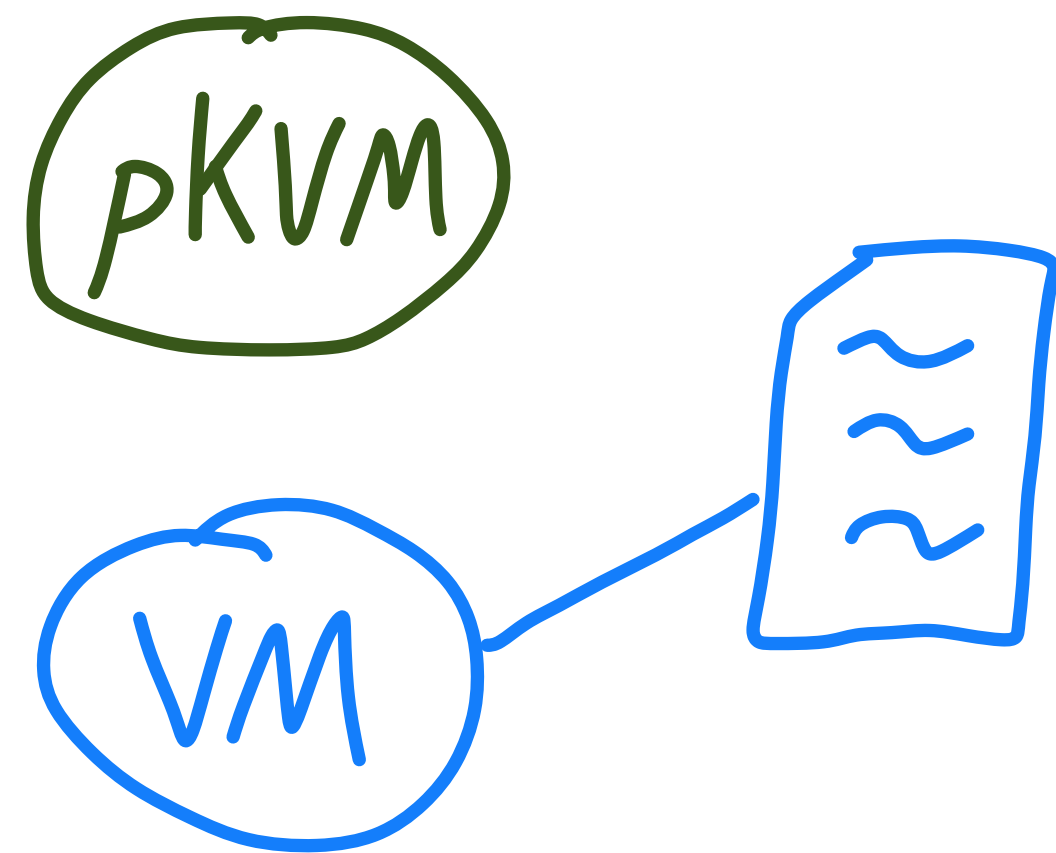
Architecture

arm

Top-down: *executable* specifications
of *pKVM*

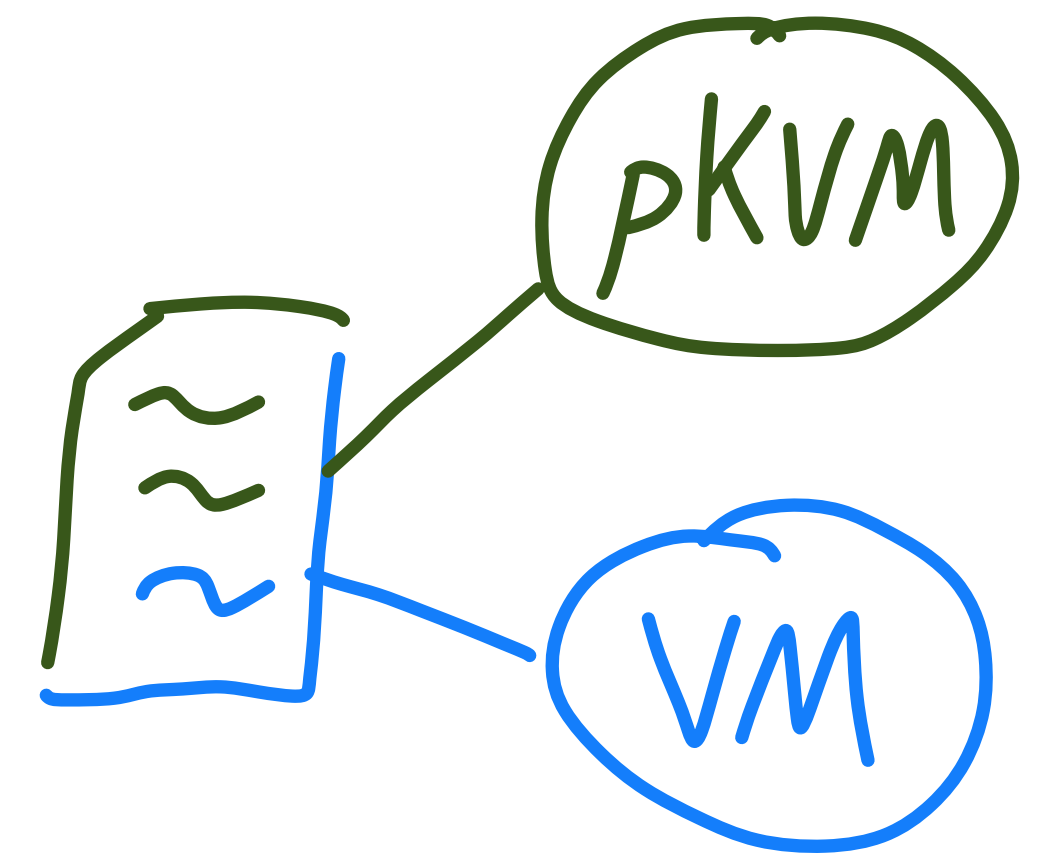
[Ghost in the Android Shell: Pragmatic Test-oracle Specification of a Production Hypervisor. SOSP 2025.
Kayvan Memarian, Ben Simner, David Kaloper-Meršinjak, Thibaut Pérami, and Peter Sewell.]

A Spec, Informally



host_share_hyp

→

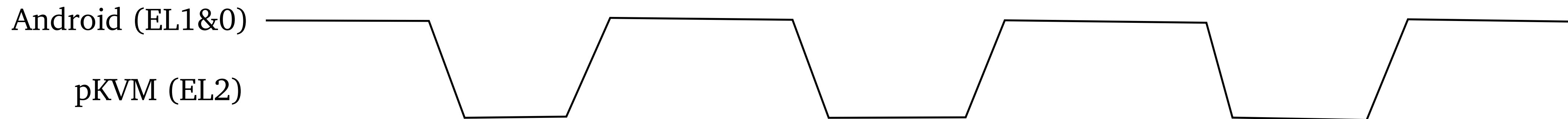


A Spec, Precisely

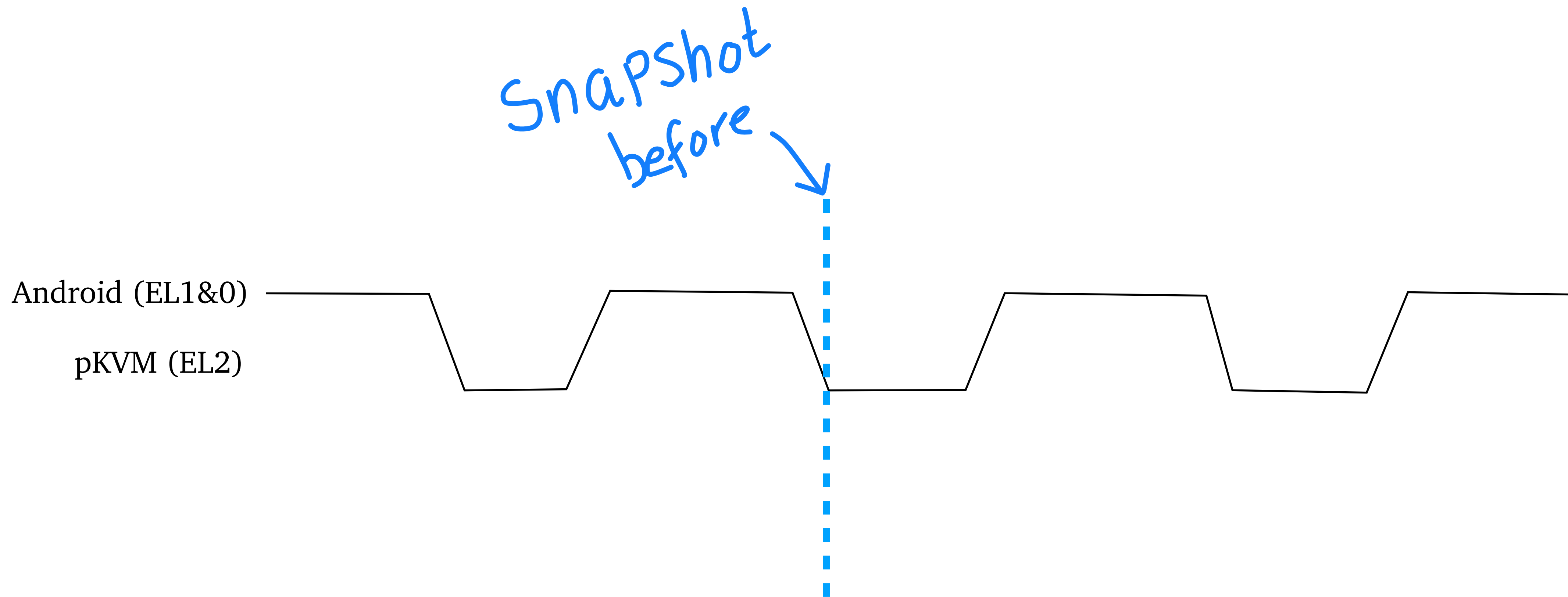
```
1 static bool compute_post__pkvm_host_share_hyp(  
2     struct ghost_state *g_post, struct ghost_state *g_pre,  
3     struct ghost_call_data *call)  
4 {  
5     // Address space conversions  
6     u64 pfn = ghost_read_gpr(g_pre, 1);  
7     phys_addr_t phys = hyp_pfn_to_phys(pfn);  
8     host_ipa_t host_addr = host_ipa_of_phys(phys);  
9     hyp_va_t hyp_addr = hyp_va_of_phys(g_pre, phys);  
0     int ret = 0;  
  
1     // Permissions checks  
2     if (!is_owned_exclusively_by(g_pre, GHOST_HOST, phys)) {  
3         ret = -EPERM;  
4         goto out;  
5     }  
  
6     // Initialisation of the (partial) post-state  
7     copy_abstraction_host(g_post, g_pre);  
8     copy_abstraction_pkvm(g_post, g_pre);  
  
9     // Construction of abstract mapping attributes  
0     bool is_memory = ghost_addr_is_allowed_memory(g_pre, phys);
```

```
24     struct maplet_attributes host_attrs =  
25         ghost_host_memory_attributes(is_memory, SHARED_OWNED);  
26     struct maplet_attributes hyp_attrs =  
27         ghost_hyp_memory_attributes(is_memory, SHARED_BORROWED);  
28  
29     // Update abstract mappings with new targets  
30     mapping_update(  
31         &g_post->host.shared,  
32         g_pre->host.shared,  
33         MAP_INSERT_PAGE, GHOST_STAGE2, host_addr, 1,  
34         maplet_target_mapped_attrs(phys, 1, host_attrs)  
35     );  
36     mapping_update(  
37         &g_post->pkvm.pgt.mapping,  
38         g_pre->pkvm.pgt.mapping,  
39         MAP_INSERT_PAGE, GHOST_STAGE1, hyp_addr, 1,  
40         maplet_target_mapped_attrs(phys, 1, hyp_attrs)  
41     );  
42  
43     // Epilogue: update the host register state  
44 out:  
45     ghost_write_gpr(g_post, 1, ret);  
46     copy_registers_to_host(g_post);  
47     return true;  
48 }
```

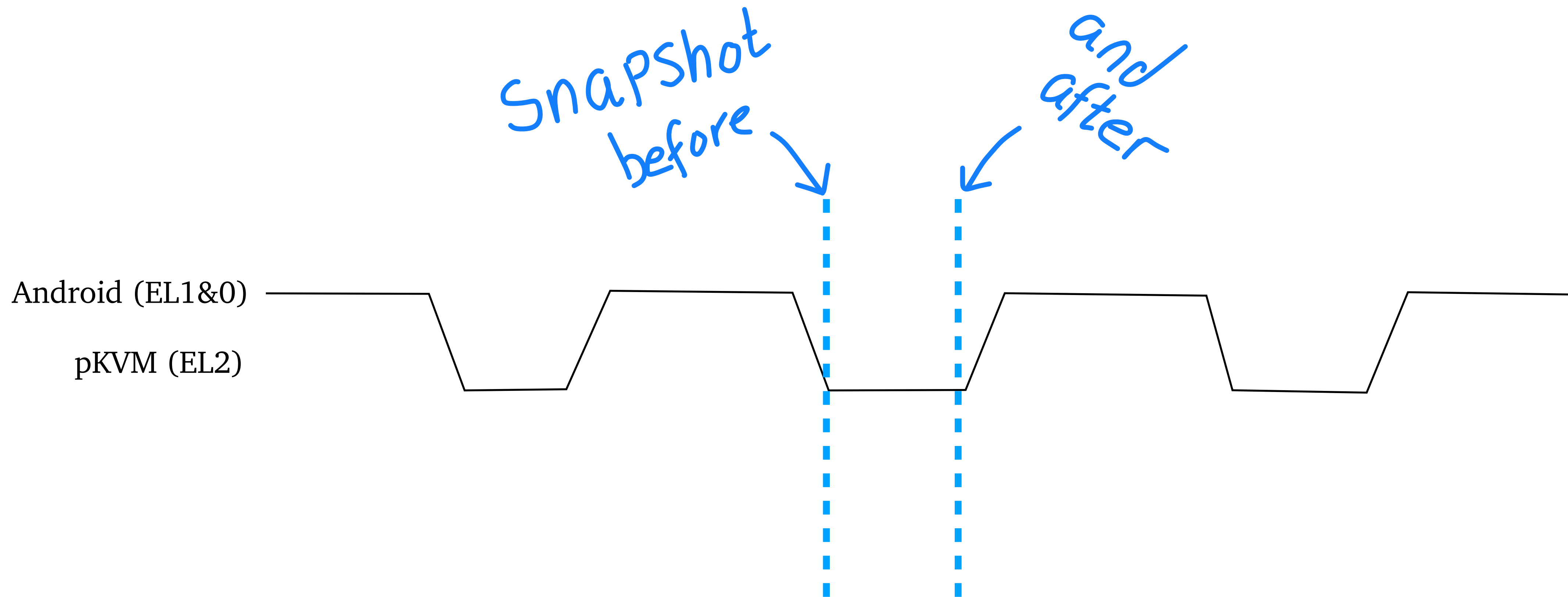
Making it executable ...



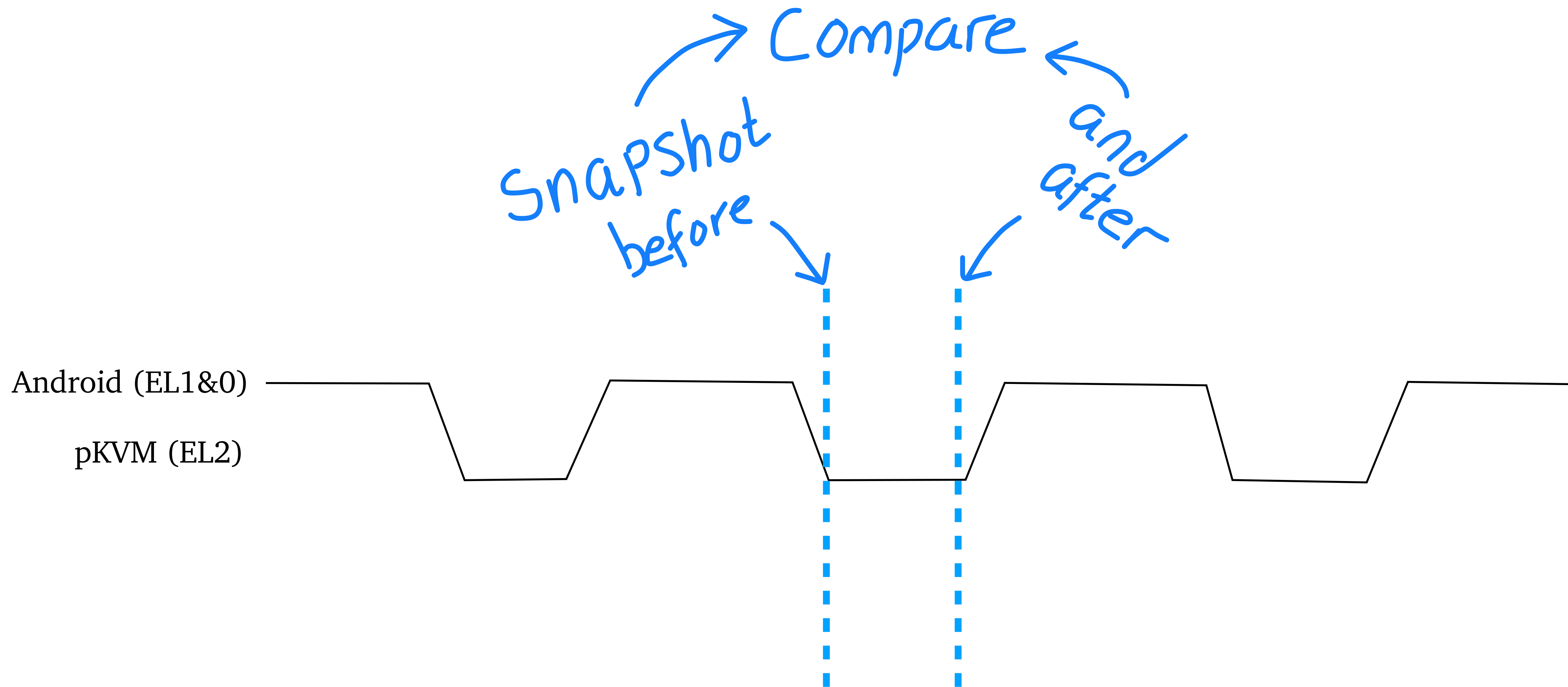
Making it executable ...



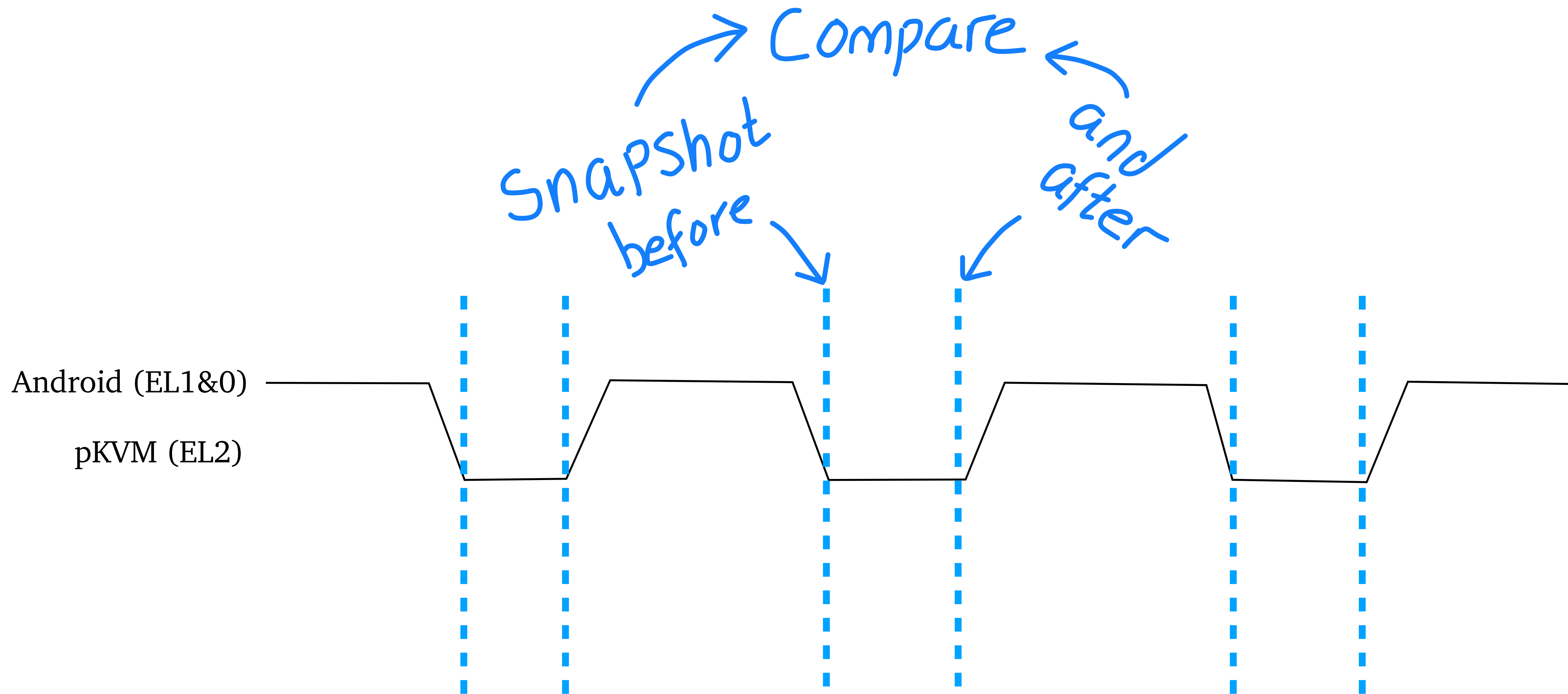
Making it executable ...



Making it executable ...

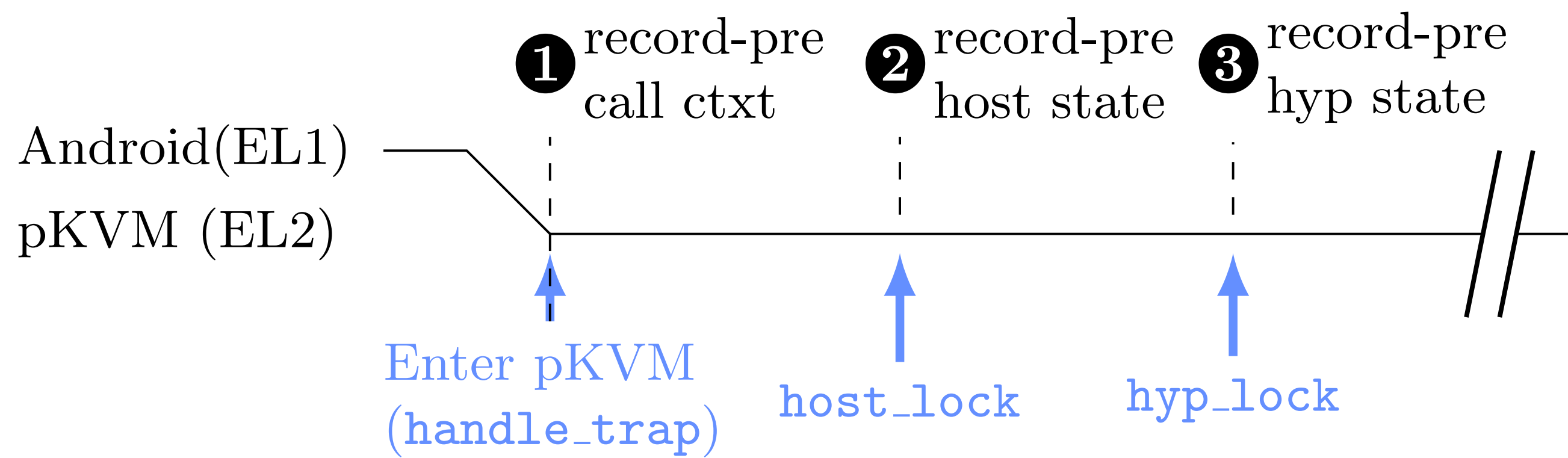


Making it executable ...

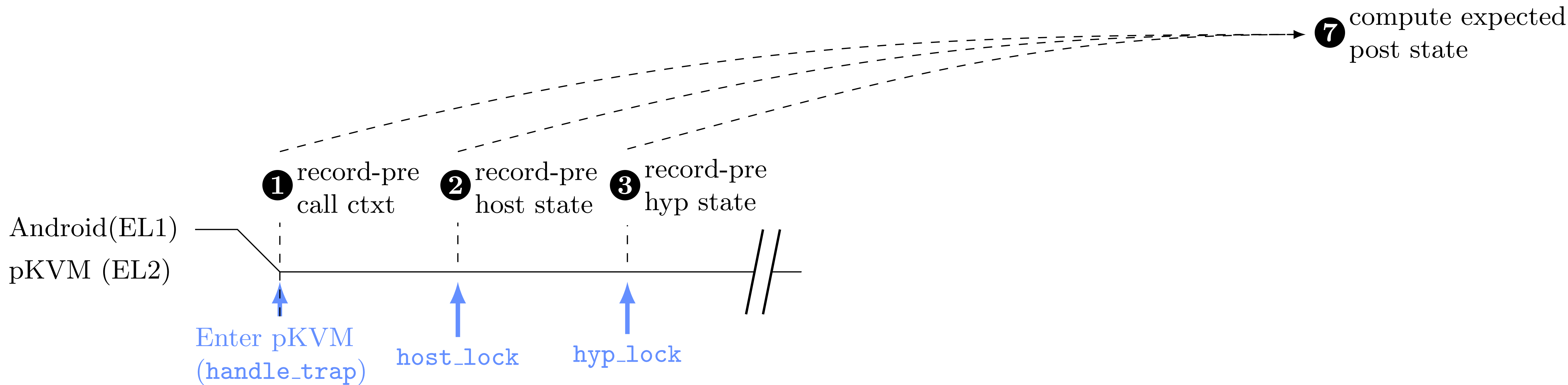


Concurrency?

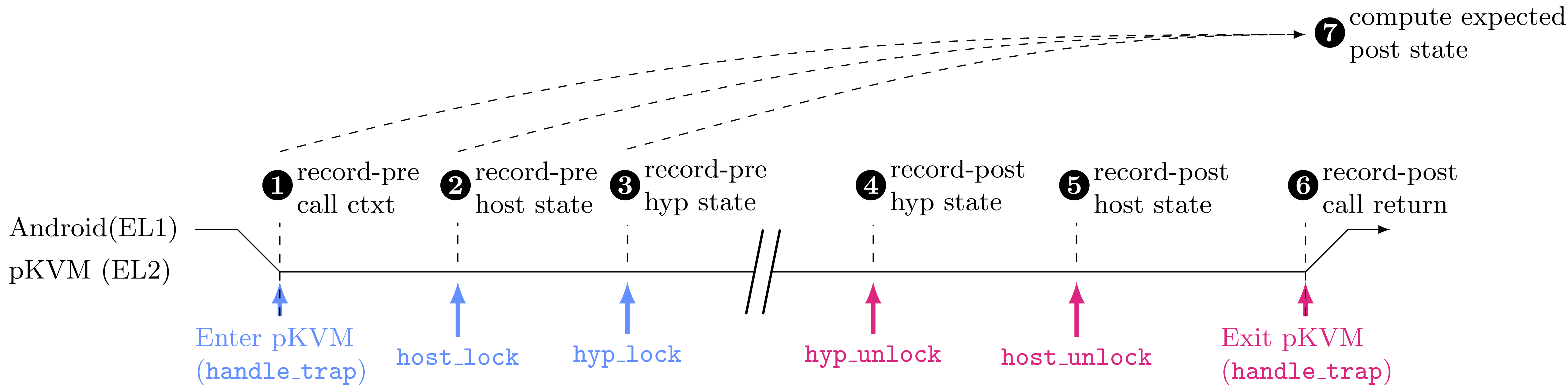
Concurrency?



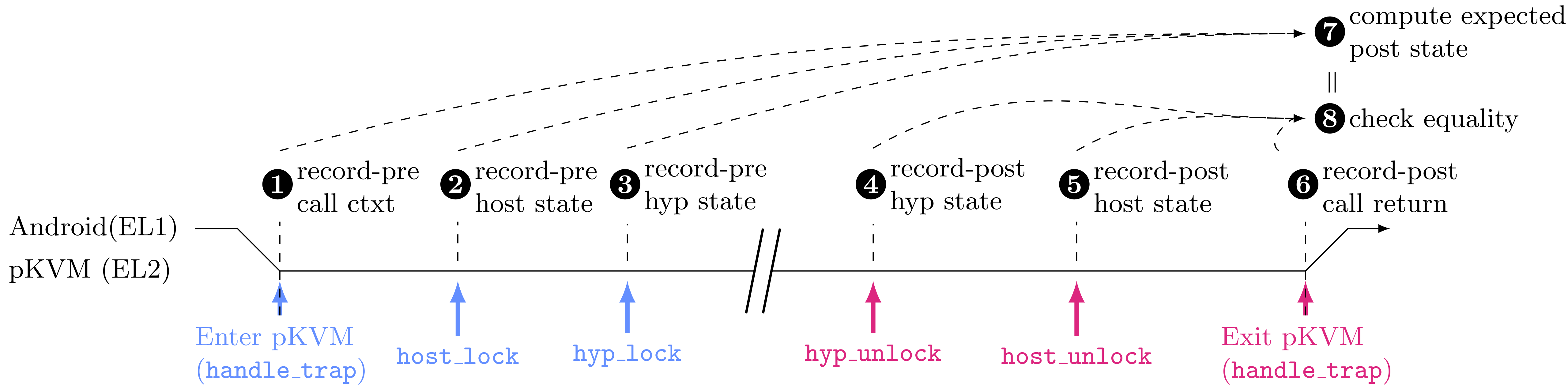
Concurrency?



Concurrency?



Concurrency?



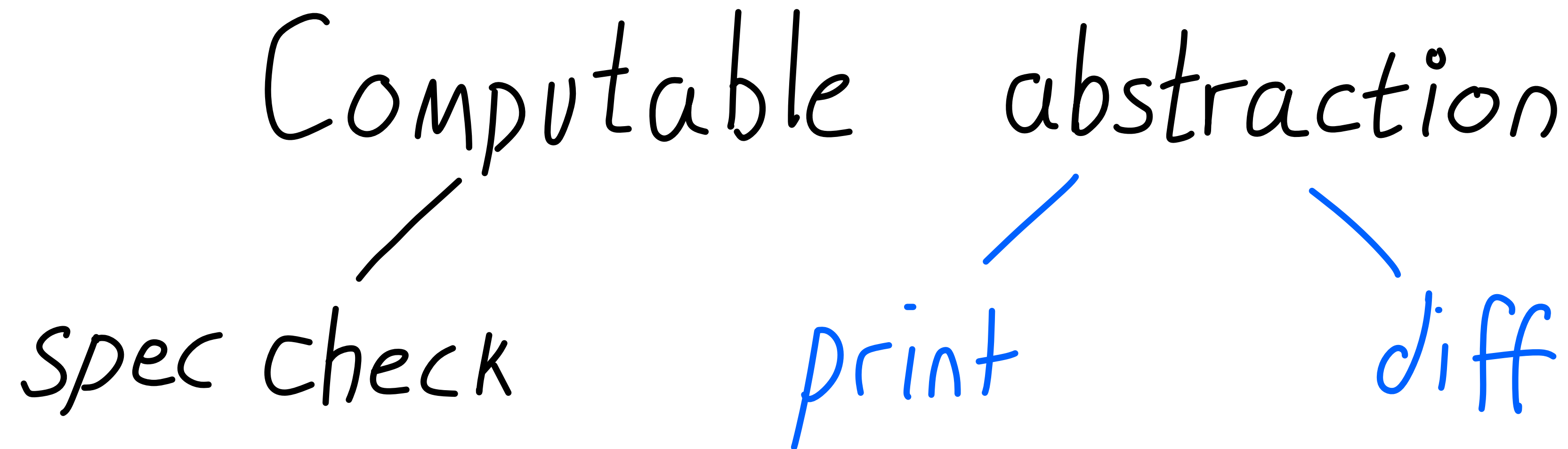
Finds bugs!

Computable abstraction

Finds bugs!

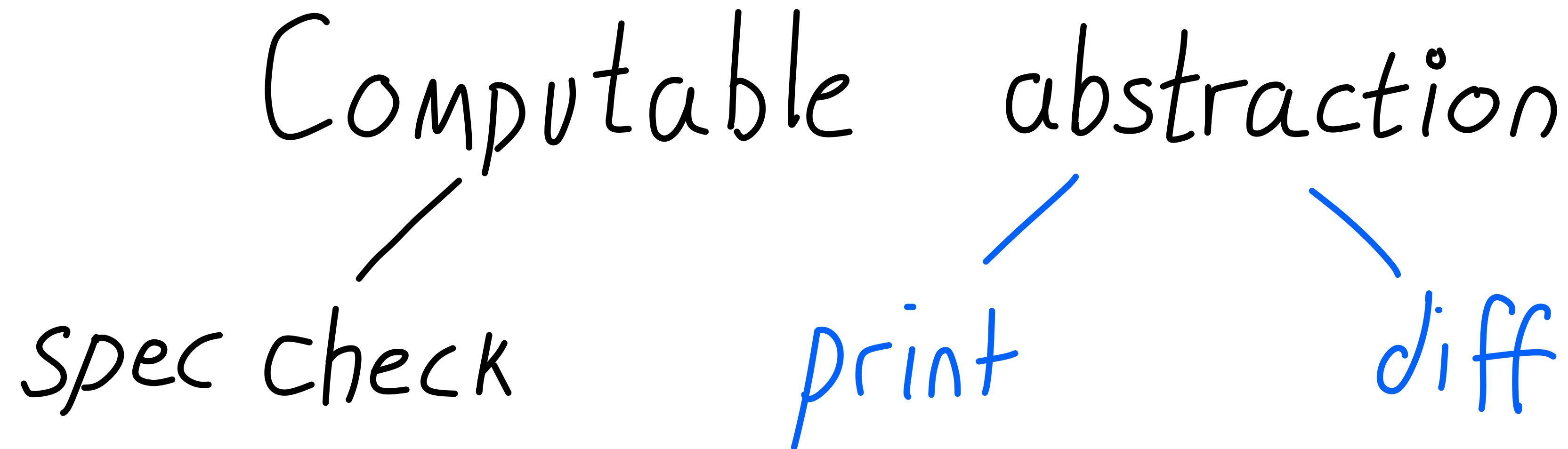
Computable abstraction
/
spec check

Finds bugs!



Finds bugs!

Just do it!



Bottom-up: caseMATE

Virtual Memory Discipline Checker

[Abstract architecture to catch concrete bugs: checking Android hypervisor TLB synchronisation. Preprint. Ben Simner, Thomas Fourier, Yeji Han, David Kaloper Meršinjak, Thibaut Pérami, Peter Sewell, and Jean Pichon-Pharabod.]

Free BSD

```
bool
vmppmap_enter(vm_offset_t va, vm_size_t size, vm_paddr_t pa, vm_prot_t prot)
{
    pd_entry_t l3e, *l3;

    l3e = /* ... */

    while (size > 0) {
        l3 = vmppmap_l3_table(va);
        if (l3 == NULL)
            return (false);

        atomic_store_64(&l3[pmap_l3_index(va)], l3e | pa);

        size -= PAGE_SIZE;
        pa += PAGE_SIZE;
        va += PAGE_SIZE;
    }

    return (true);
}
```

```
static pd_entry_t *
vmppmap_l3_table(vm_offset_t va)
{
    /* ... */
    l2e = atomic_load_64(&l2[pmap_l2_index(va)]);
    if ((l2e & ATTR_DESCR_VALID) == 0) {
        /* Allocate a page for the level 3 table */
        m = vm_page_alloc_noobj(VM_ALLOC_WIRED | VM_ALLOC_ZERO);
        new_l2e = VM_PAGE_TO_PHYS(m) | L2_TABLE;
        mtx_lock(&vmppmap_mtx);
        rv = atomic_cmpset_64(&l2[pmap_l2_index(va)], l2e, new_l2e);
        mtx_unlock(&vmppmap_mtx);
        /* We may have raced another thread, try again */
        if (rv == 0)
            /* ... */
            /* The cmpset succeeded */
            l2e = new_l2e;
    }
    l3 = (pd_entry_t *)PHYS_TO_DMAP(l2e & ~ATTR_MASK);
    return (l3);
}
```

Free BSD

```
bool
vmppmap_enter(vm_offset_t va, vm_size_t size, vm_paddr_t pa, vm_prot_t prot)
{
    pd_entry_t l3e, *l3;

    l3e = /* ... */

    while (size > 0) {
        l3 = vmppmap_l3_table(va);
        if (l3 == NULL)
            return (false);

        atomic_store_64(&l3[pmap_l3_index(va)], l3e | pa);

        size -= PAGE_SIZE;
        pa += PAGE_SIZE;
        va += PAGE_SIZE;
    }

    return (true);
}
```

```
static pd_entry_t *
vmppmap_l3_table(vm_offset_t va)
{
    /* ... */
    l2e = atomic_load_64(&l2[pmap_l2_index(va)]);
    if ((l2e & ATTR_DESCR_VALID) == 0) {
        /* Allocate a page for the level 3 table */
        m = vm_page_alloc_noobj(VM_ALLOC_WIRED | VM_ALLOC_ZERO);
        new_l2e = VM_PAGE_TO_PHYS(m) | L2_TABLE;
        mtx_lock(&vmppmap_mtx);
        rv = atomic_cmpset_64(&l2[pmap_l2_index(va)], l2e, new_l2e);
        mtx_unlock(&vmppmap_mtx);
        /* We may have raced another thread, try again */
        if (rv == 0)
            /* ... */
            /* The cmpset succeeded */
            l2e = new_l2e;
    }
    l3 = (pd_entry_t *)PHYS_TO_DMAP(l2e & ~ATTR_MASK);
    return (l3);
}
```

Free BSD

```
bool
vmppmap_enter(vm_offset_t va, vm_size_t size, vm_paddr_t pa, vm_prot_t prot)
{
    pd_entry_t l3e, *l3;

    l3e = /* ... */

    while (size > 0) {
        l3 = vmppmap_l3_table(va);
        if (l3 == NULL)
            return (false);

        atomic_store_64(&l3[pmap_l3_index(va)], l3e | pa);

        size -= PAGE_SIZE;
        pa += PAGE_SIZE;
        va += PAGE_SIZE;
    }

    return (true);
}
```

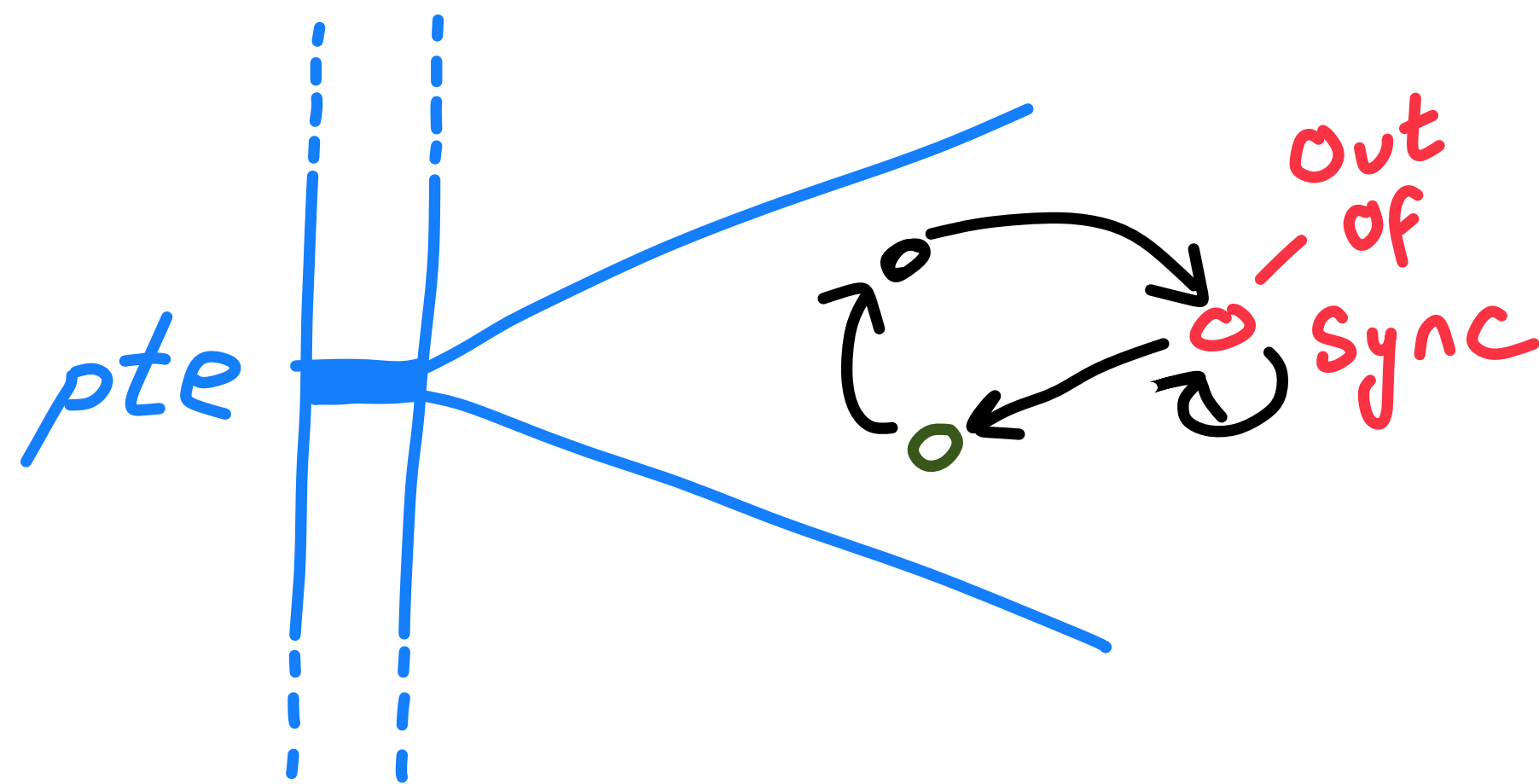
```
static pd_entry_t *
vmppmap_l3_table(vm_offset_t va)
{
    /* ... */
    l2e = atomic_load_64(&l2[pmap_l2_index(va)]);
    if ((l2e & ATTR_DESCR_VALID) == 0) {
        /* Allocate a page for the level 3 table */
        m = vm_page_alloc_noobj(VM_ALLOC_WIRED | VM_ALLOC_ZERO);
        new_l2e = VM_PAGE_TO_PHYS(m) | L2_TABLE;
        mtx_lock(&vmppmap_mtx);
        rv = atomic_cmpset_64(&l2[pmap_l2_index(va)], l2e, new_l2e);
        mtx_unlock(&vmppmap_mtx);
        /* We may have raced another thread, try again */
        if (rv == 0)
            /* ... */
            /* The cmpset succeeded */
            l2e = new_l2e;
    }
    l3 = (pd_entry_t *)PHYS_TO_DMAP(l2e & ~ATTR_MASK);
    return (l3);
}
```

Racy with the H/W



Casemate

Pageable update protocol



As sound abstraction



CaseMATE

Dynamic checking

```
mtx_lock(&vmppmap_mtx);
rv = atomic_cmpset_64(&l0[pmap_l0_index(va)], l0e, new_l0e);
#if defined(__CASEMATE_FREEBSD__)
casemate_model_step_lock((uint64_t)&vmppmap_mtx);
if (rv) {
    /* CASEMATE: success, wrote l0e */
    uint64_t __phys = DMAP_TO_PHYS((uint64_t)&l0[pmap_l0_index(va)]);
    casemate_model_step_write(WM0_plain, __phys, l0e);
}
casemate_model_step_unlock((uint64_t)&vmppmap_mtx);
#endif
mtx_unlock(&vmppmap_mtx);
```



CaseMATE

Dynamic checking

```
mtx_lock(&vmppmap_mtx);
rv = atomic_cmpset_64(&l0[pmap_l0_index(va)], l0e, new_l0e);
#if defined(__CASEMATE_FREEBSD__)
casemate_model_step_lock((uint64_t)&vmppmap_mtx);
if (rv) {
    /* CASEMATE: success, wrote l0e */
    uint64_t __phys = DMAP_TO_PHYS((uint64_t)&l0[pmap_l0_index(va)]);
    casemate_model_step_write(WM0_plain, __phys, l0e);
}
casemate_model_step_unlock((uint64_t)&vmppmap_mtx);
#endif
mtx_unlock(&vmppmap_mtx);
```

Annotate



CaseMATE

Dynamic checking

```

mtx_lock(&vmmpmap_mtx);
rv = atomic_cmpset_64(&l0[pmap_l0_index(va)], l0e, new_l0e);
#if defined(__CASEMATE_FREEBSD__)
casemate_model_step_lock((uint64_t)&vmmpmap_mtx);
if (rv) {
    /* CASEMATE: success, wrote l0e */
    uint64_t __phys = DMAP_TO_PHYS((uint64_t)&l0[pmap_l0_index(va)]);
    casemate_model_step_write(WM0_plain, __phys, l0e);
}
casemate_model_step_unlock((uint64_t)&vmmpmap_mtx);
#endif
mtx_unlock(&vmmpmap_mtx);

```

Annotate

```

***** TRAP (from host) *****
__pkvm_host_share_hyp
[r1] pfn: 0x.....1372e2
W 0x.....eabf9000 0x.....0 at arch/arm64/kvm/hyp/nvhe/page_alloc.c:84 in page_remove_from_list
W 0x.....eabf9008 0x.....0 at arch/arm64/kvm/hyp/nvhe/page_alloc.c:84 in page_remove_from_list
W 0x.....eaa8edc8 0x.....400 at arch/arm64/kvm/hyp/nvhe/./pgtable.c:968 in stage2_try_set_pte
transition simplified model state diff:
mem:
- *[0x.....eaa8edc8]=0x.....1372007fd (pte_st:V root:0x.....eaa1b000)
+ *[0x.....eaa8edc8]=0x.....400 (pte_st:IU n 0 root:0x.....eaa1b000)
Wrel 0x.....eaa8edc8 0x.....eabf9003 at arch/arm64/kvm/hyp/nvhe/./pgtable.c:1040 in stage2_make_pte
! BBM invalid unclean->valid
[ 162.133816] kvm [241]: nVHE hyp BUG at: arch/arm64/kvm/hyp/nvhe/ghost_simplified_model.c:1045!

```

Report violations



CaseMATE

Dynamic checking

```

mtx_lock(&vmmpmap_mtx);
rv = atomic_cmpset_64(&l0[pmap_l0_index(va)], l0e, new_l0e);
#if defined(__CASEMATE_FREEBSD__)
casemate_model_step_lock((uint64_t)&vmmpmap_mtx);
if (rv) {
    /* CASEMATE: success, wrote l0e */
    uint64_t __phys = DMAP_TO_PHYS((uint64_t)&l0[pmap_l0_index(va)]);
    casemate_model_step_write(WM0_plain, __phys, l0e);
}
casemate_model_step_unlock((uint64_t)&vmmpmap_mtx);
#endif
mtx_unlock(&vmmpmap_mtx);

```

Violations = Security-critical bugs!

Annotate

```

***** TRAP (from host) *****
__pkvm_host_share_hyp
[r1] pfn: 0x.....1372e2
W 0x.....eabf9000 0x.....0 at arch/arm64/kvm/hyp/nvhe/page_alloc.c:84 in page_remove_from_list
W 0x.....eabf9008 0x.....0 at arch/arm64/kvm/hyp/nvhe/page_alloc.c:84 in page_remove_from_list
W 0x.....eaa8edc8 0x.....400 at arch/arm64/kvm/hyp/nvhe/./pgtable.c:968 in stage2_try_set_pte
transition simplified model state diff:
mem:
- *[0x.....eaa8edc8]=0x.....1372007fd (pte_st:V root:0x.....eaa1b000)
+ *[0x.....eaa8edc8]=0x.....400 (pte_st:IU n 0 root:0x.....eaa1b000)
Wrel 0x.....eaa8edc8 0x.....eabf9003 at arch/arm64/kvm/hyp/nvhe/./pgtable.c:1040 in stage2_make_pte
! BBM invalid unclean->valid
[ 162.133816] kvm [241]: nVHE hyp BUG at: arch/arm64/kvm/hyp/nvhe/ghost_simplified_model.c:1045!

```

Report violations

Lessons!

- Specs are easy to write
... and useful!

Lessons!

- Specs are easy to write
... and useful!
- H/W protocols can be checked
... and s/w buggy!

Lessons!

- Specs are easy to write
... and useful!
- H/W protocols can be checked
... and s/w buggy!

~~Thanks!~~