Virtual Machine Boot-up Analysis

Abderrahmane Benbachir

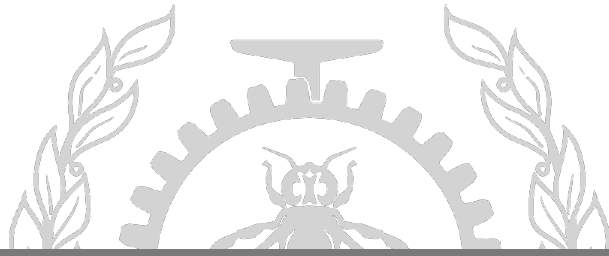
Dec 12, 2016

École Polytechnique de Montréal

Laboratoire **DORSAL**

Agenda

- Introduction
- Research objectives
- Investigation
- Results
- Future work

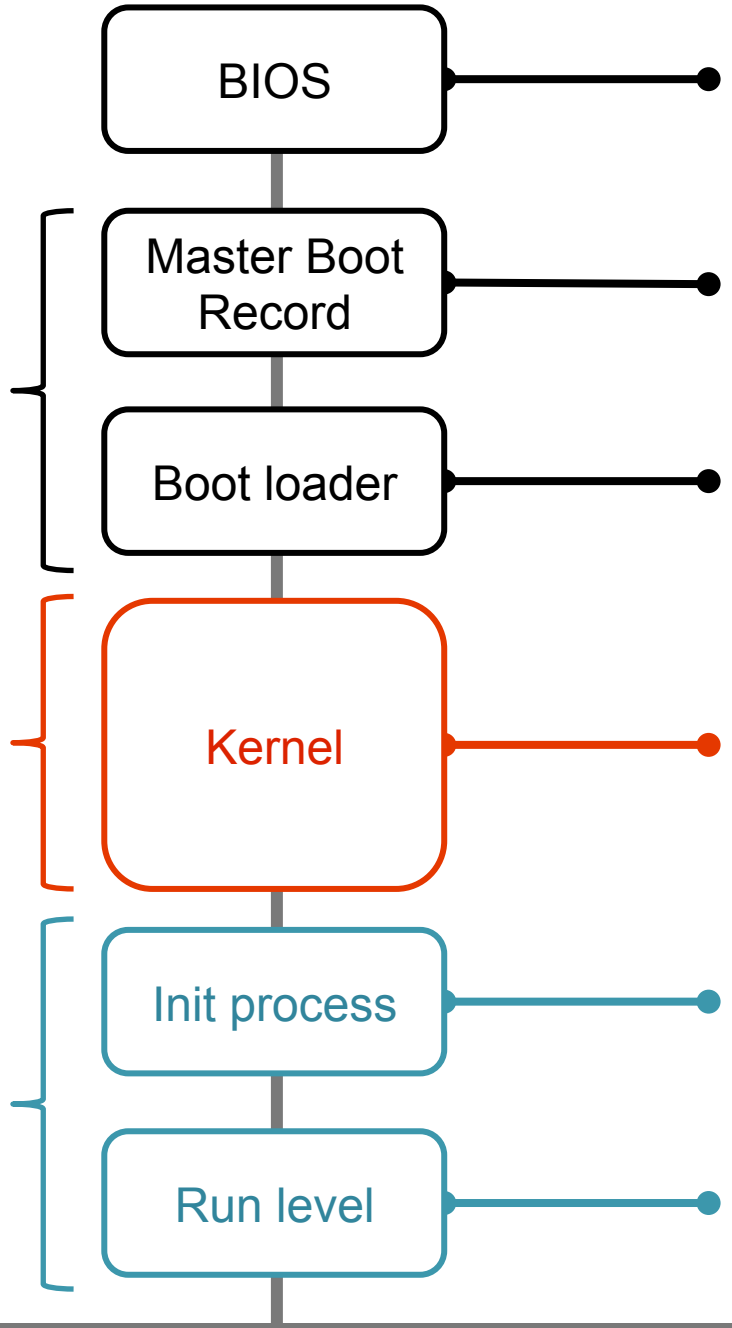


Introduction

Still a dark place

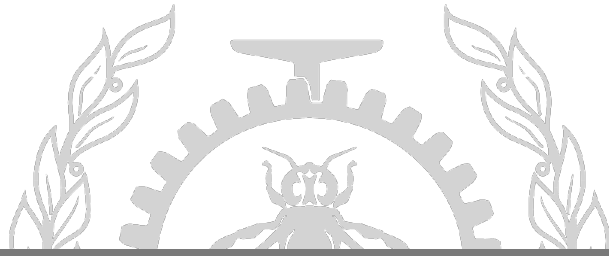
Ftrace partially handle this part

LTTng, **Ftrace**, **Dtrace**, ...



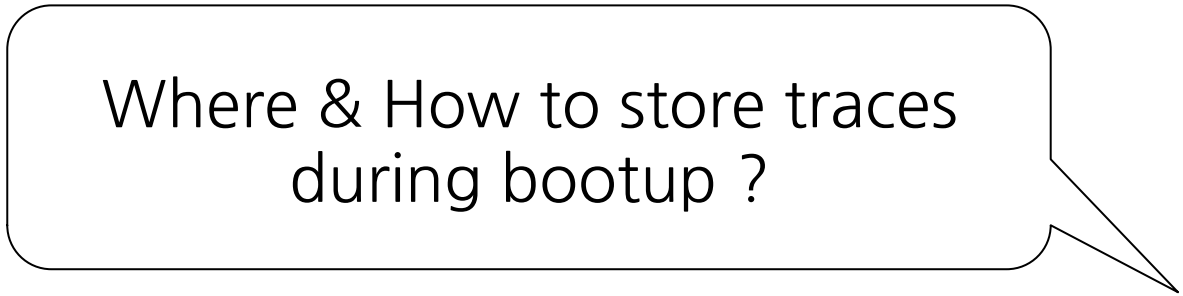
Research objectives

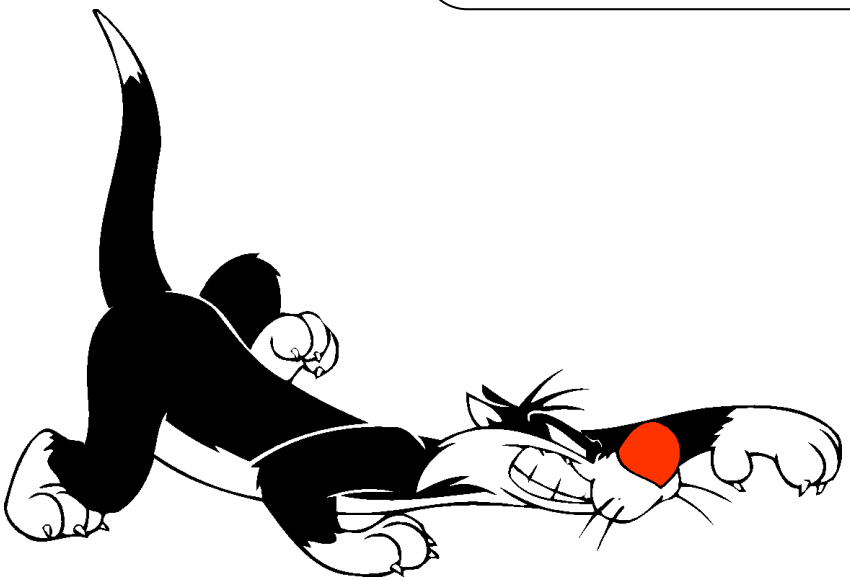
- Analyze virtual machines early boot-up
- Detect boot up issues



Investigation

We already know how to trace: **LTTng**, **Ftrace**, **Dtrace**, **Perf**.





Source : https://s-media-cache-ak0.pinimg.com/736x/99/66/28/996628fa4292de7175d7893ca31d7796.jpg

Where & How to store traces during boot up ?

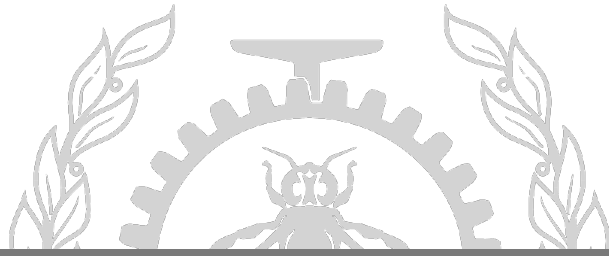
Keep them inside **Guest**

- Recording Timestamp
- Synchronization
- Buffer size
- Memory or Disk

VS

Offload them to the **Host**

- Guest or Host Timestamp ?
- Synchronization
- Transfer Channels (Network, Shared Memory, Paravirt)
- Transfer Speed $$$



Investigation

Where & How to store traces during boot up ?

Keep them inside **Guest** vs Offload them to the **Host**

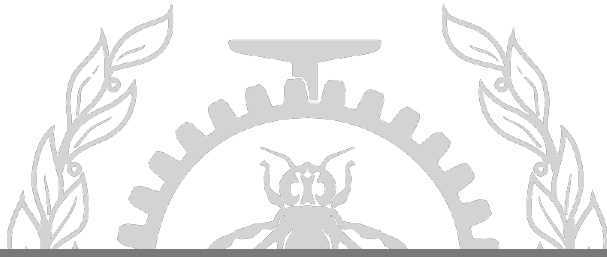
Light weight Tracers

Network : Sockets

Shared Memory: Qemu Hypertrace

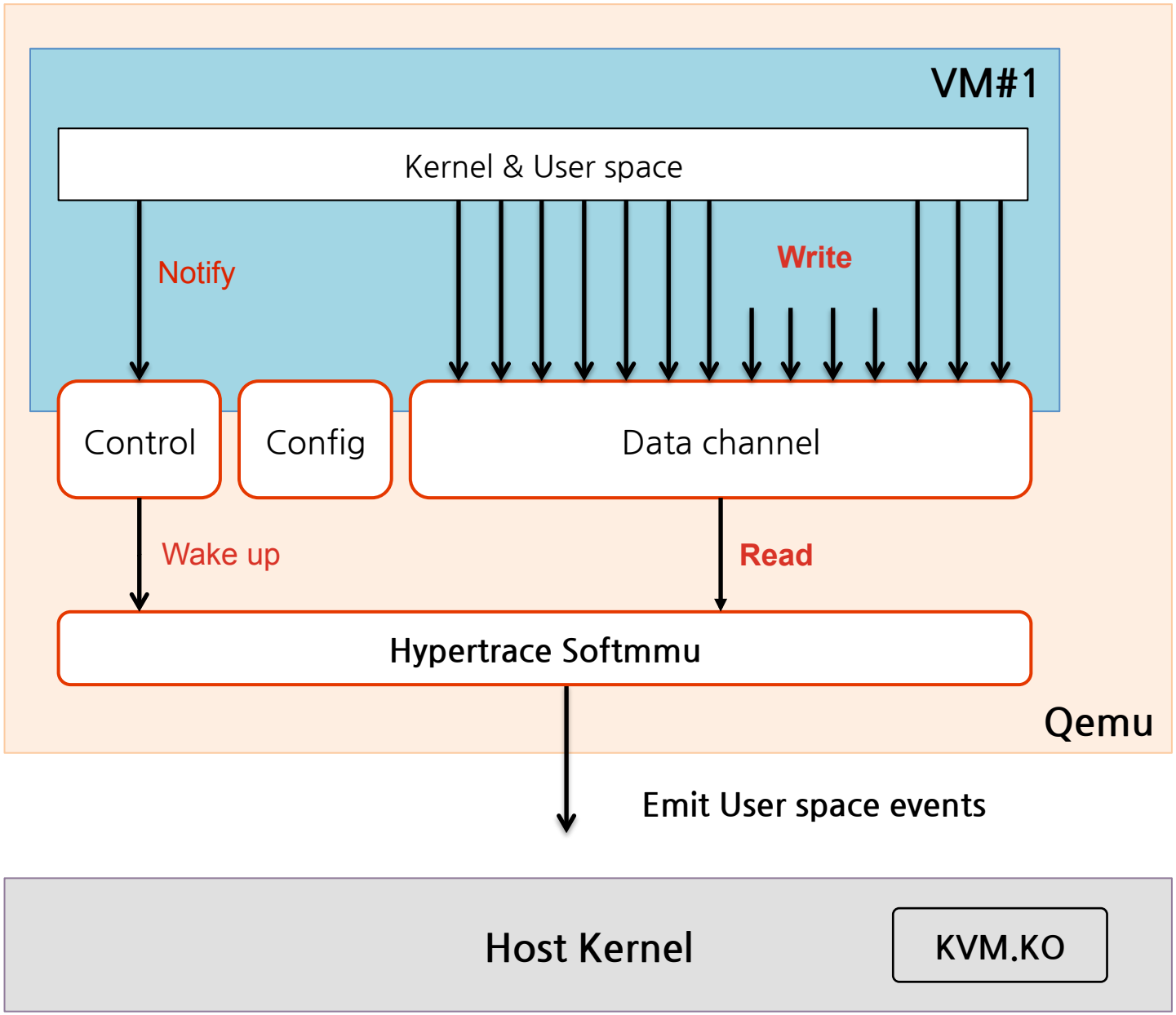
Paravirtualization API: Hypercalls

Must be explored



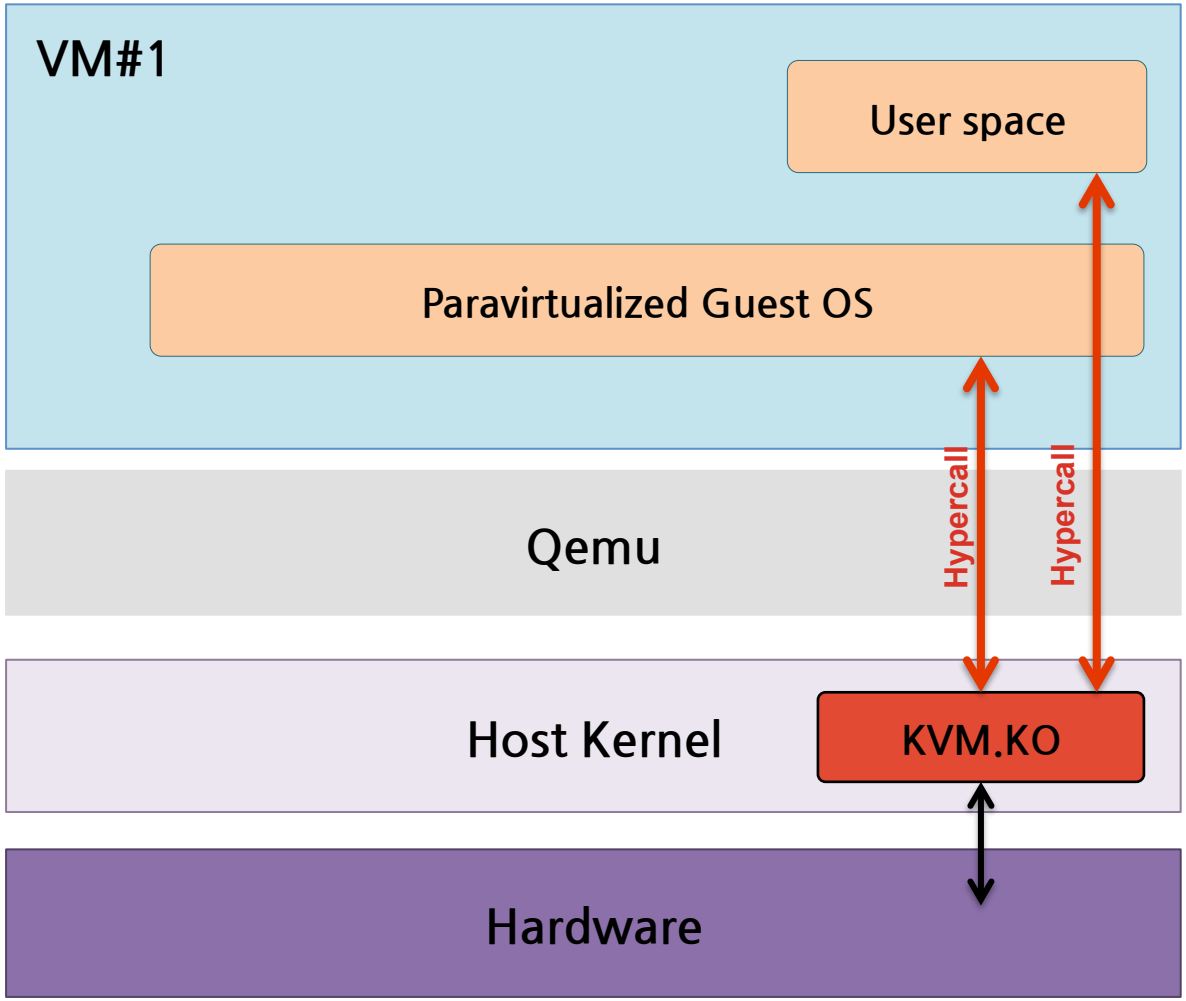
Qemu Hypertrace

By Lluís Vilanova



Shared Memory Btw Guest & Qemu

Hypercall



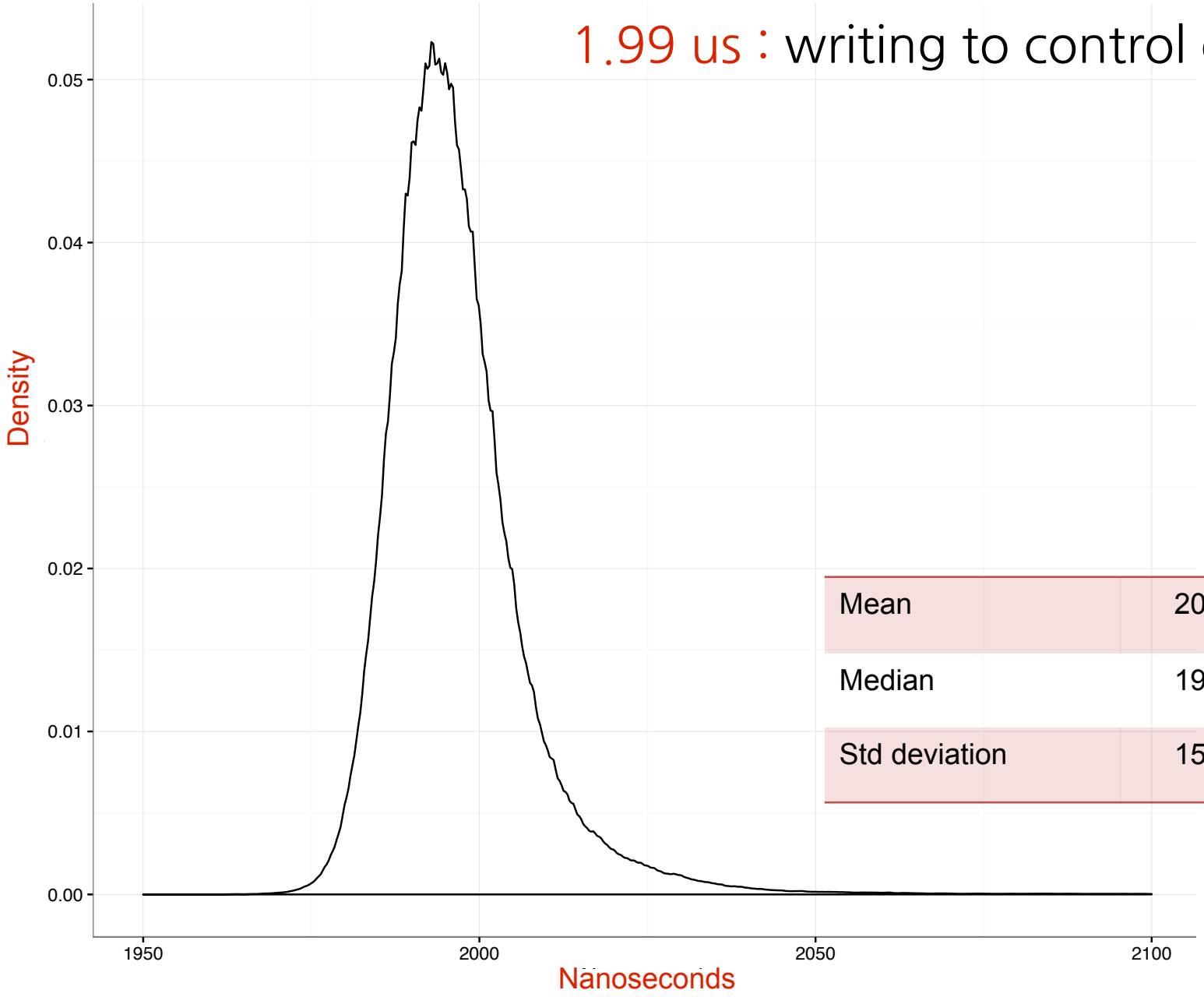
Direct Execution of Guest Requests

This layer is not involved

Benchmarks

Qemu Hypertrace Overhead (KVM Enabled)

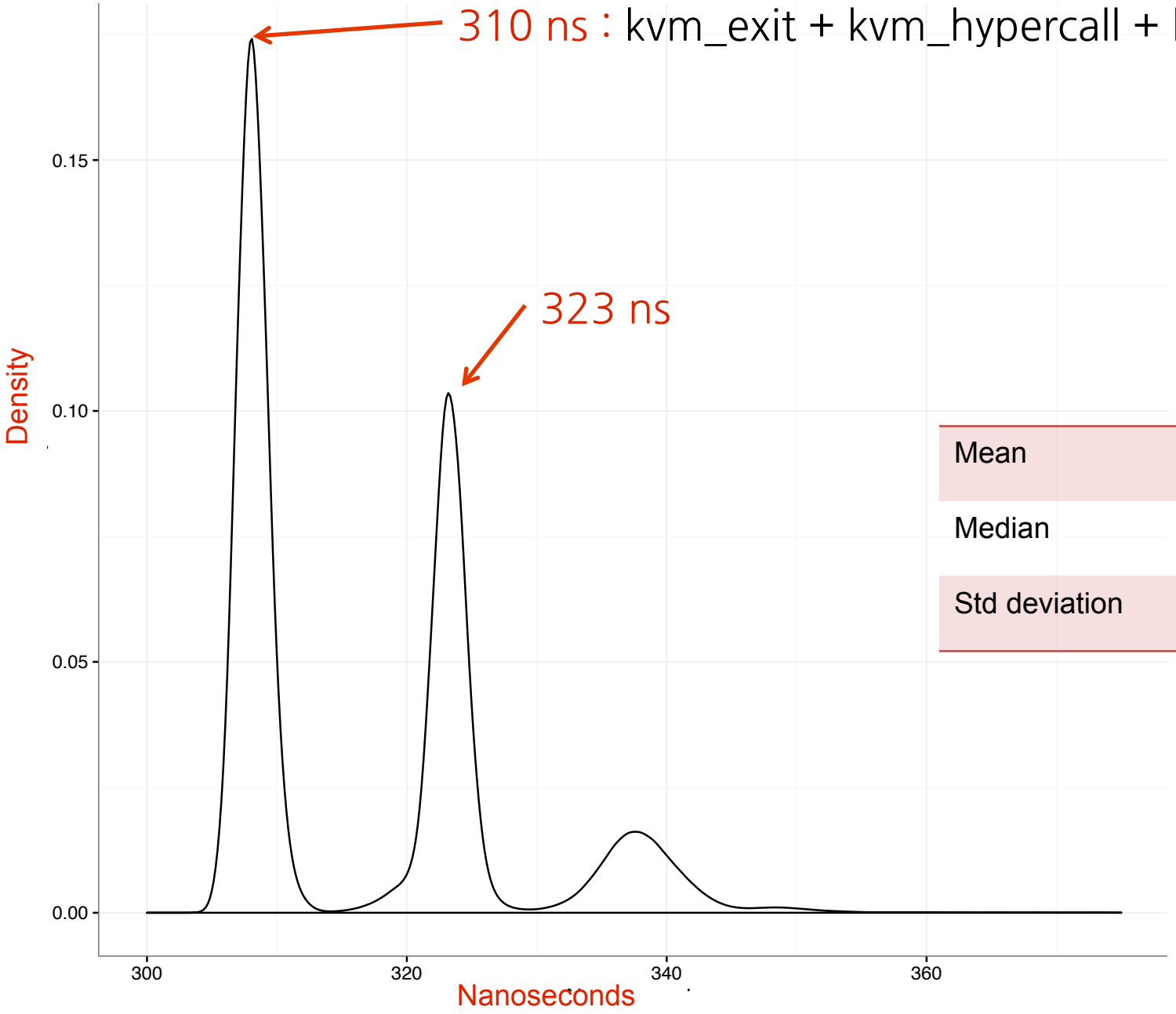
1.99 us : writing to control channel



Mean	2070 ns
Median	1995 ns
Std deviation	1580 ns

Benchmarks

Hypercall Overhead

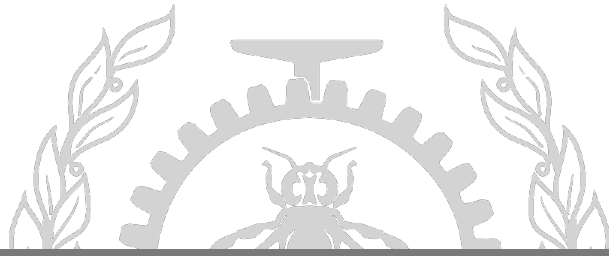


Mean	340 ns
Median	310 ns
Std deviation	833 ns

Hypercalls Implementation

How to trace through Hypercalls

- Hook to Ftrace “function graph” callback for entry & exit
- Only trace the Host
- Use only host Timestamp : No synchronization required
- Dump kernel symbols to map each function names



Hypercalls Implementation

hypercall payload = 5 args x 64 bits = 40 Bytes

hypercall(nr, a0, a1, a2, a3)

Hypercall number

Entry

Exit

[13:59:00.035571626] (+0.000000494) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { **nr** = 1000, **a0** = **18446744071579284592**, **a1** = **0**, **a2** = 0, **a3** = 0 }

[13:59:00.035572670] (+0.000000519) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579288208, a1 = 1, a2 = 4198, a3 = 3 }
[13:59:00.035573178] (+0.000000508) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579292656, a1 = 1, a2 = 5227, a3 = 2 }
[13:59:00.035573686] (+0.000000508) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579922416, a1 = 1, a2 = 6251, a3 = 1 }
[13:59:39.016994097] (+0.000000482) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4197127, a1 = 0, a2 = 0, a3 = 0 }
[13:59:00.035584321] (+0.000000531) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579265552, a1 = 0, a2 = 0, a3 = 0 }
[13:59:00.035586871] (+0.000000490) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579284528, a1 = 0, a2 = 0, a3 = 0 }

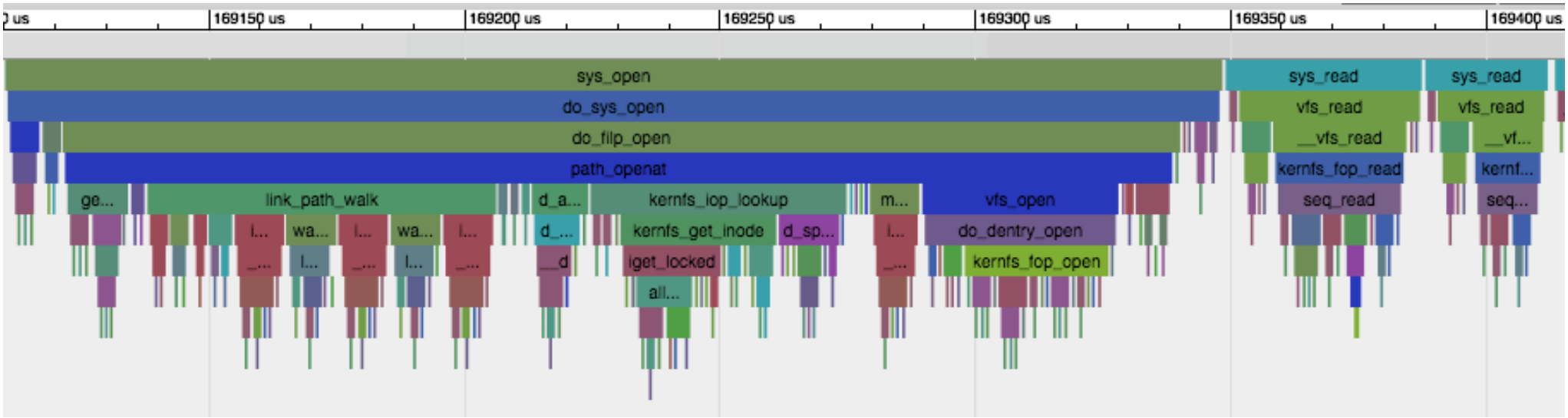
[13:59:00.035587383] (+0.000000525) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { **nr** = 1000, **a0** = **18446744071579284592**, **a1** = **1**, **a2** = 40, **a3** = 4 }

[13:59:39.016994570] (+0.000000473) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4197127, a1 = 1, a2 = 0, a3 = 0 }
[13:59:39.016995047] (+0.000000477) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4196150, a1 = 0, a2 = 0, a3 = 0 }
[13:59:39.016996002] (+0.000000481) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4196518, a1 = 0, a2 = 0, a3 = 0 }
[13:59:00.035585361] (+0.000000503) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579284592, a1 = 0, a2 = 0, a3 = 0 }
[13:59:00.035585863] (+0.000000502) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071587317584, a1 = 0, a2 = 0, a3 = 0 }
[13:59:39.017128748] (+0.000000734) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4196518, a1 = 1, a2 = 0, a3 = 0 }
[13:59:39.017129250] (+0.000000502) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4197127, a1 = 1, a2 = 0, a3 = 0 }

Function graph

Chrome Trace-Viewer

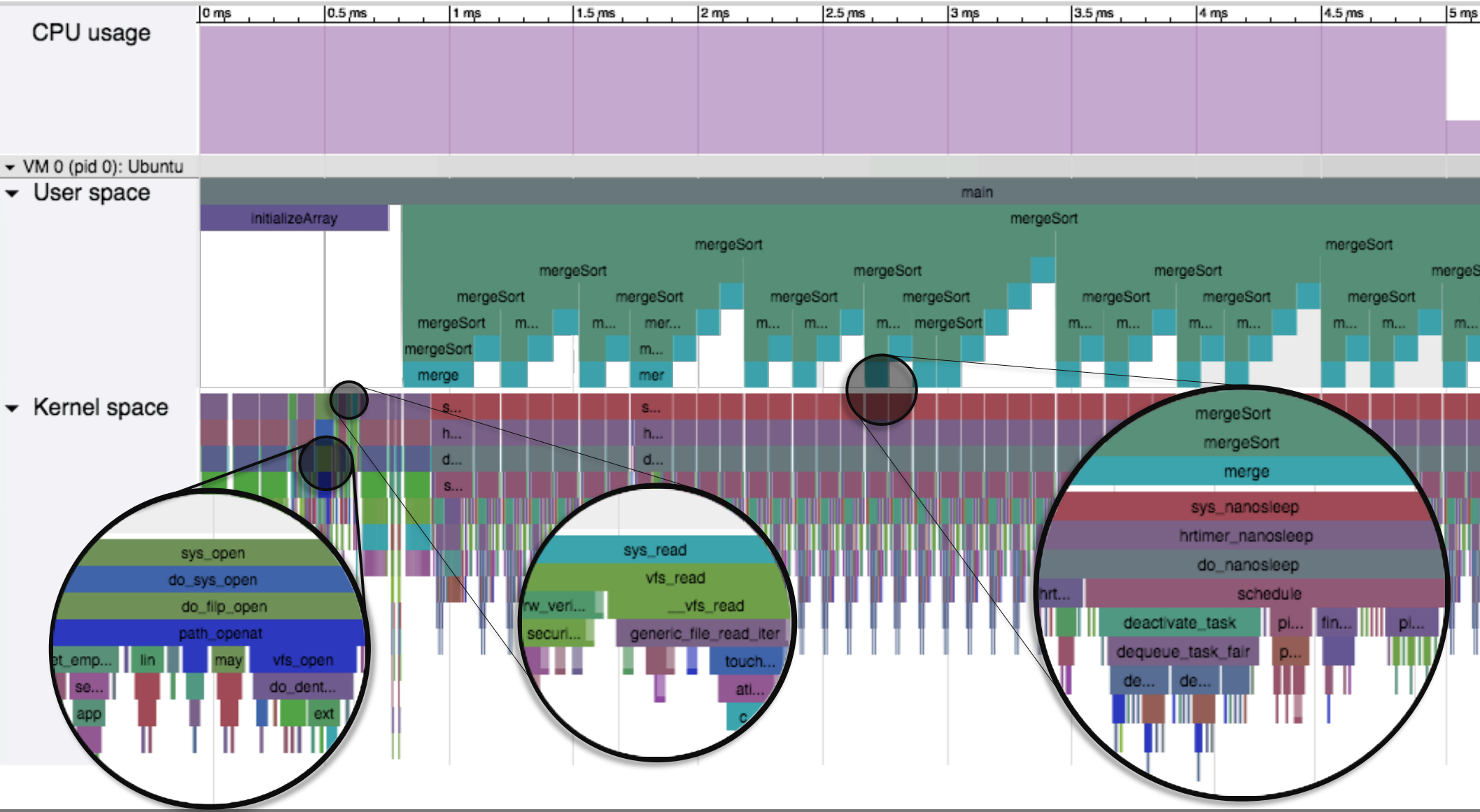
Callstacks of Guest Kernel space



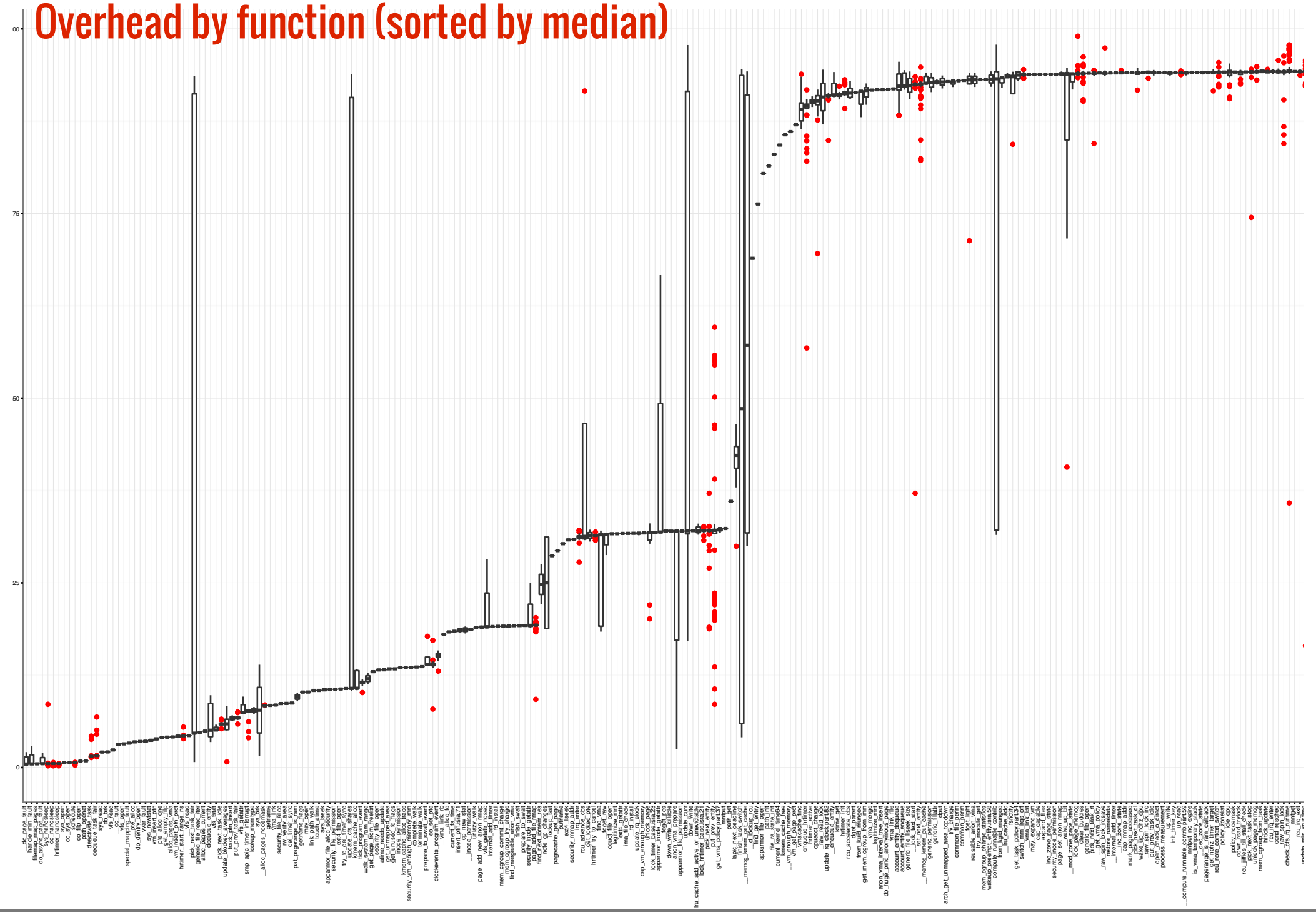
300 us

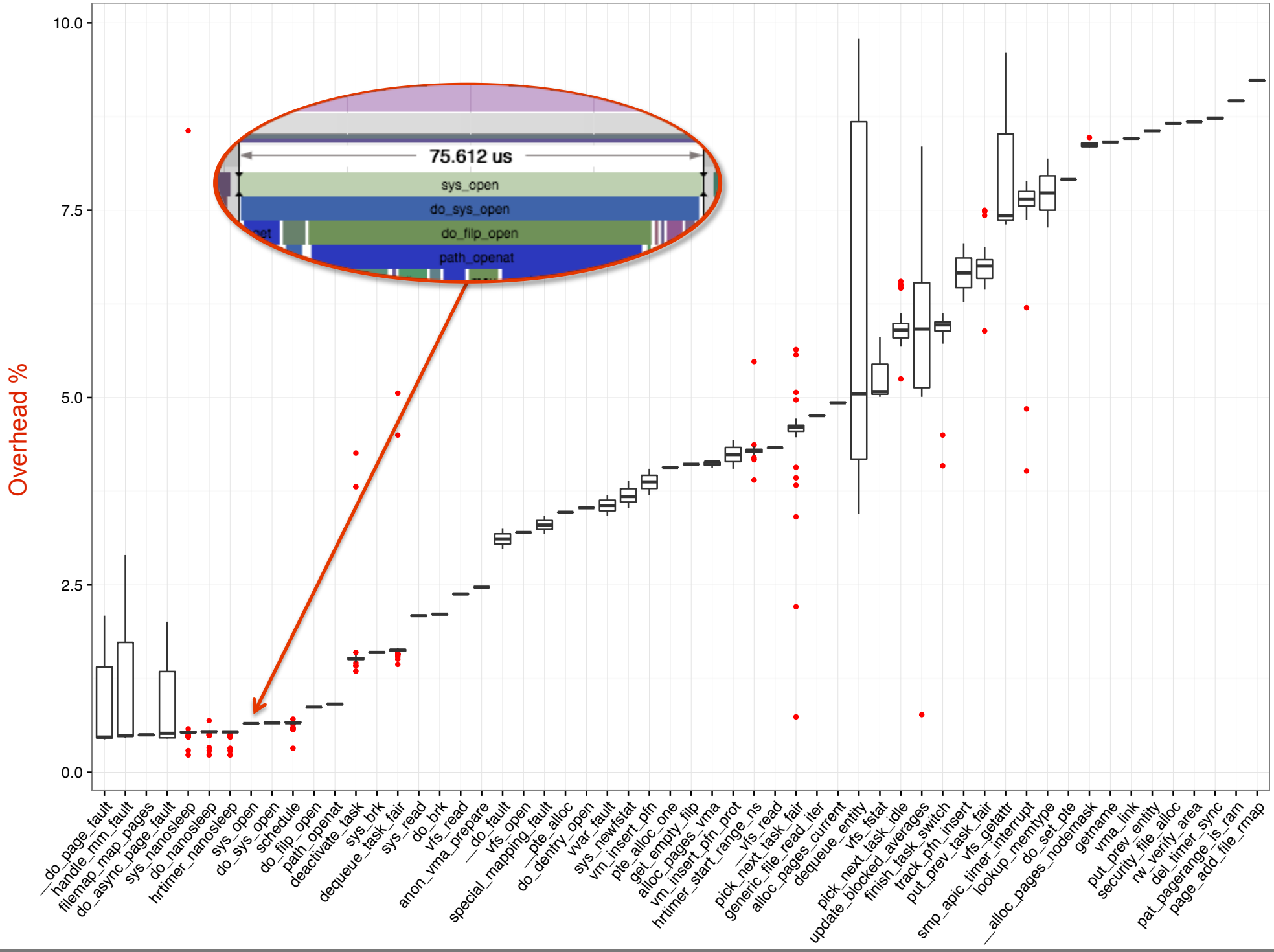
Hybrid tracing technique

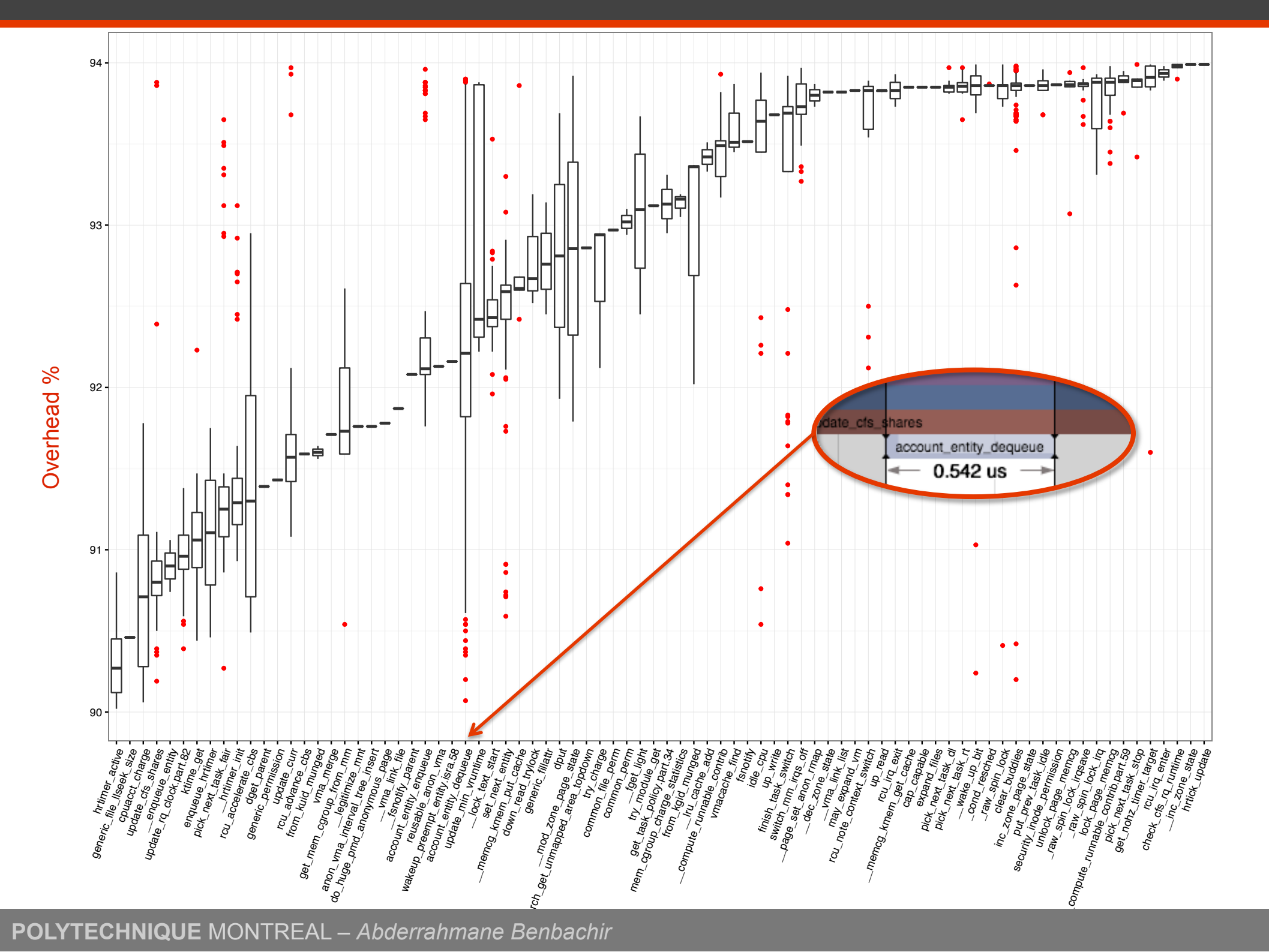
Kernel space + User space



Overhead by function (sorted by median)

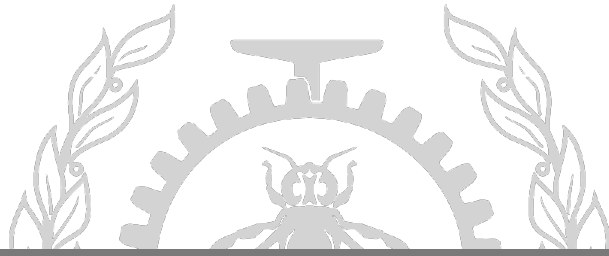



Overhead %
10.0
7.5
5.0
2.5
0.0
75.612 us
sys_open
do_sys_open
do_filp_open
path_openat
__do_page_fault
handle_mm_fault
filemap_map_pages
do_async_page_fault
sys_nanosleep
do_nanosleep
hrtimer_nanosleep
sys_open
do_sys_open
schedule
do_filp_open
path_openat
deactivate_task
sys_brk
dequeue_task_fair
sys_read
do_brk
vfs_read
anon_vma_prepare
__do_fault
vfs_open
special_mapping_fault
__pte_alloc
do_dentry_open
vvar_fault
sys_newfstat
vm_insert_pfn
pte_alloc_one
get_empty_filp
alloc_pages_vma
vm_insert_pfn_prot
hrtimer_start_range_ns
__vfs_read
pick_next_task_fair
generic_file_read_iter
alloc_pages_current
dequeue_entity
vfs_fstat
pick_next_task_idle
update_blocked_averages
finish_task_switch
track_pfn_insert
put_prev_task_fair
vfs_getattr
smp_apic_timer_interrupt
lookup_memtype
do_set_pte
__alloc_pages_nodemask
getname
vma_link
put_prev_entity
security_file_alloc
rw_verify_area
del_timer_sync
pat_pagerange_is_ram
page_add_file_rmap


Overhead %
94
93
92
91
90
update_cfs_shares
account_entity_dequeue
0.542 us
hrtimer_active
generic_file_llseek_size
cpuacct_charge
update_cfs_shares
__enqueue_entity
update_rq_clock.part.82
ktime_get
enqueue_hrtimer
pick_next_task_fair
__hrtimer_init
rcu_accelerate_cbs
dget_parent
generic_permission
update_curr
rcu_advance_cbs
from_kuid_munged
vma_merge
get_mem_cgroup_from_mm
__legitimize_mnt
anon_vma_interval_tree_insert
do_huge_pmd_anonymous_page
__vma_link_file
__fsnotify_parent
account_entity_enqueue
reusable_anon_vma
wakeup_preempt_entity.isra.58
account_entity_dequeue
update_min_vruntime
__lock_text_start
set_next_entity
__memcg_kmem_put_cache
down_read_trylock
generic_fillattr
__mod_zone_page_state
dput
arch_get_unmapped_area_topdown
try_charge
common_file_perm
common_perm
__fget_light
try_module_get
get_task_policy.part.34
mem_cgroup_charge_statistics
from_kgid_munged
__lru_cache_add
__compute_runnable_contrib
vmacache_find
fsnotify
idle_cpu
up_write
finish_task_switch
switch_mm_irqs_off
__page_set_anon_rmap
__dec_zone_state
__vma_link_list
may_expand_vm
rcu_note_context_switch
up_read
rcu_irq_exit
__memcg_kmem_get_cache
cap_capable
expand_files
pick_next_task_dl
pick_next_task_rt
__wake_up_bit
__cond_resched
_raw_spin_lock
clear_buddies
inc_zone_page_state
put_prev_task_idle
security_inode_permission
unlock_page_memcg
_raw_spin_lock_irqsave
_raw_spin_lock_irq
lock_page_memcg
compute_runnable_contrib.part.59
pick_next_task_stop
get_nohz_timer_target
rcu_irq_enter
check_cfs_rq_runtime
__inc_zone_state
hrtick_update

Future Work

- Apply hypercall technique to trace full kernel boot up
- Explore this solution with nested virtual machine
- Can we trace the Boot loader too ?



Questions

Abderrahmane.benbachir@polymtl.ca

https://github.com/abenbachir

Thanks to Professor Michel Dagenais and our partners EfficiOS and Ericsson.

References

Hypercall Implementation : https://gist.github.com/abenbachir/344822b5ba9fc5ac384cdec3f087e018

QEMU Hypertrace Patches: http://patchwork.ozlabs.org/project/qemu-devel/list/?state=&q=Hypertrace&archive