Host-Assisted Virtual Machine Tracing and Analysis

Abderrahmane Benbachir
Michel Dagenais

Dec 7, 2017

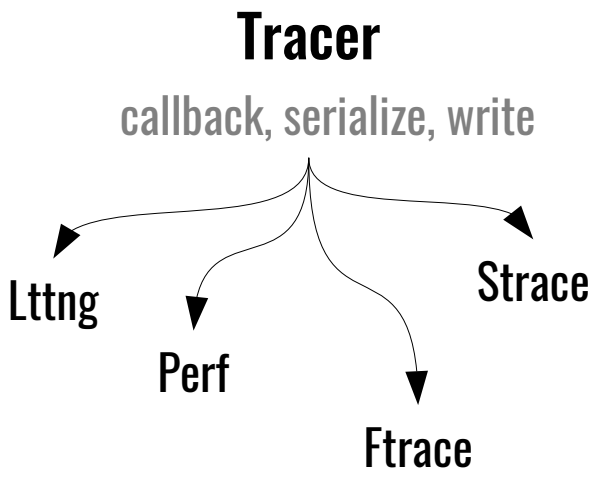
École Polytechnique de Montréal

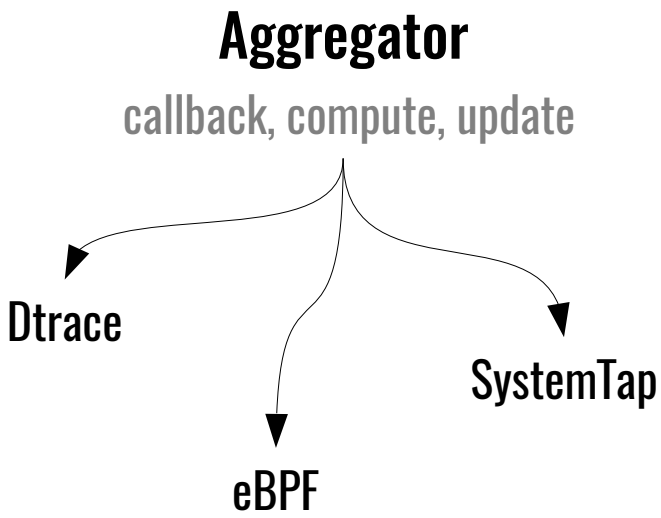
Laboratoire **DORSAL**

Agenda

- Introduction
- Hypertracing
  - Hypercall
    - Boot-up Analysis
  - Shared Memory
    - Virtualization Awareness
- Future Work
- Questions

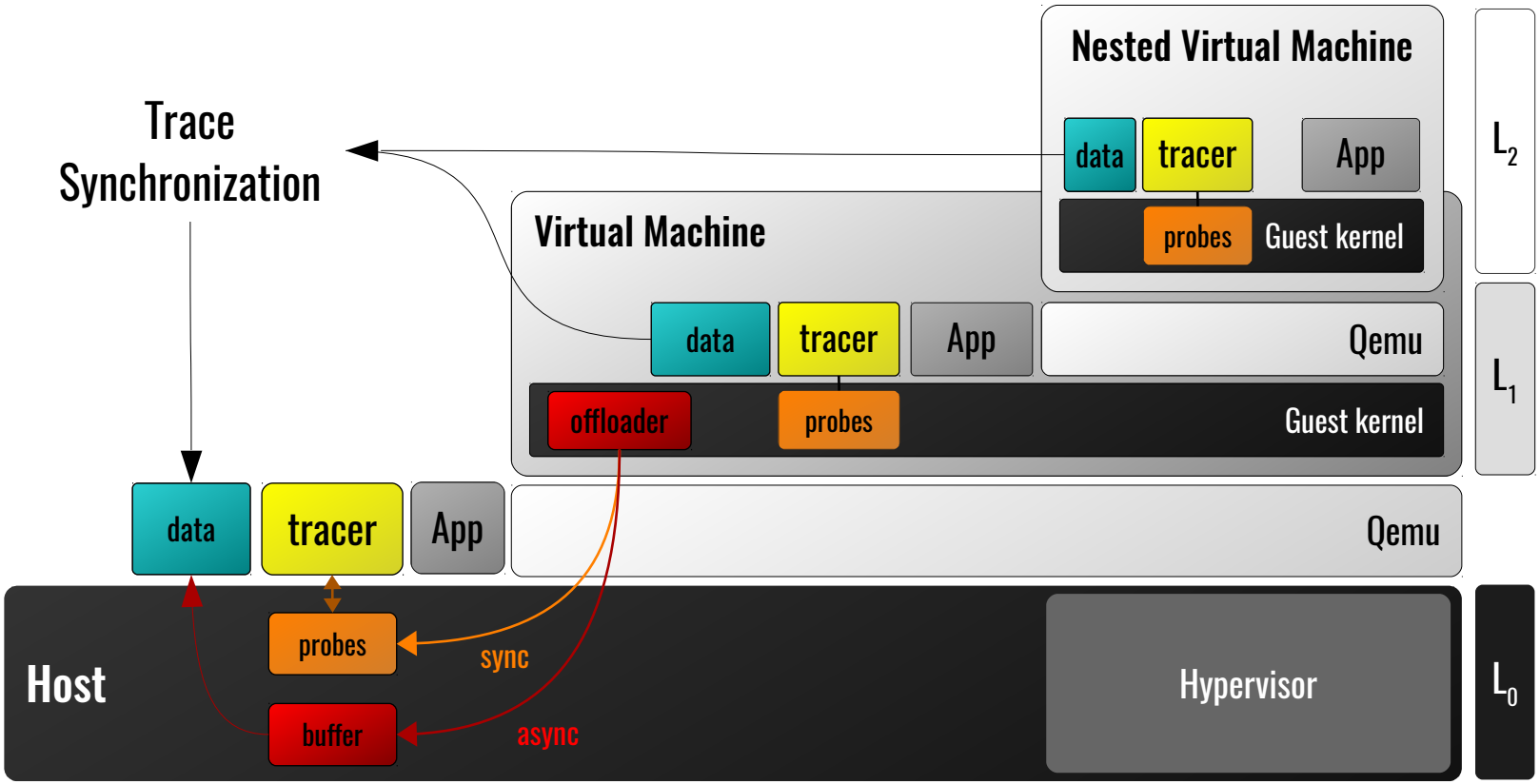
Monitoring Tools



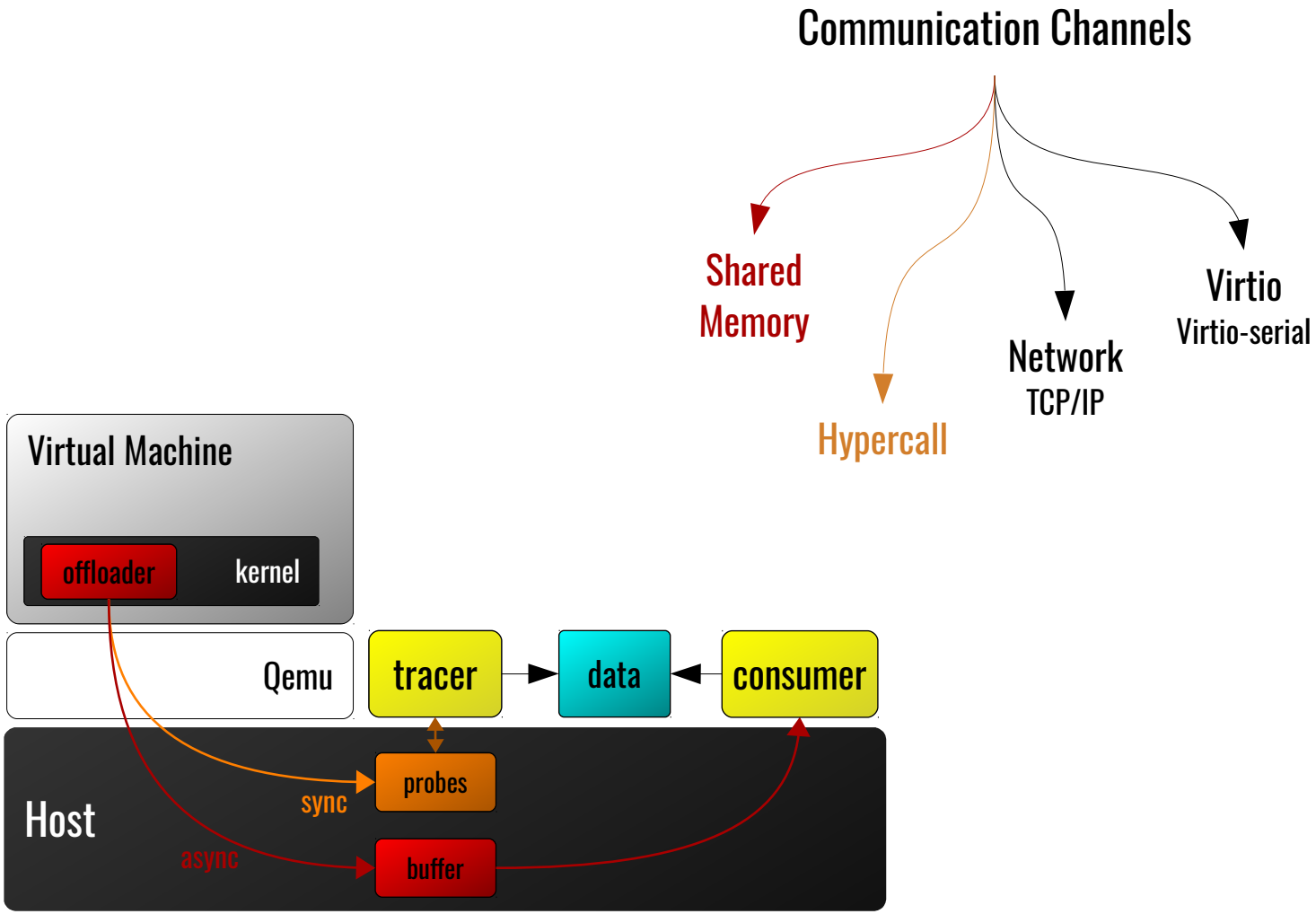


What is hypertracing?

Hypertracing = Offloading traces from guest to host (or vice versa)

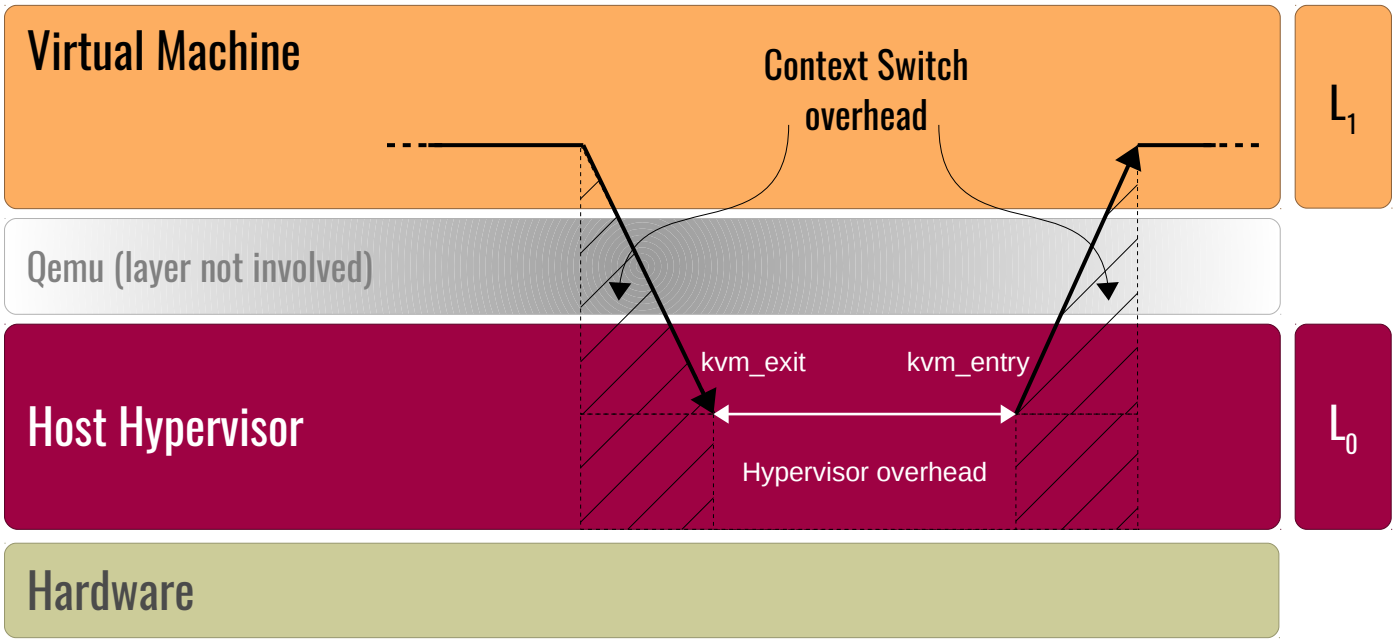


How to do hypertracing ?



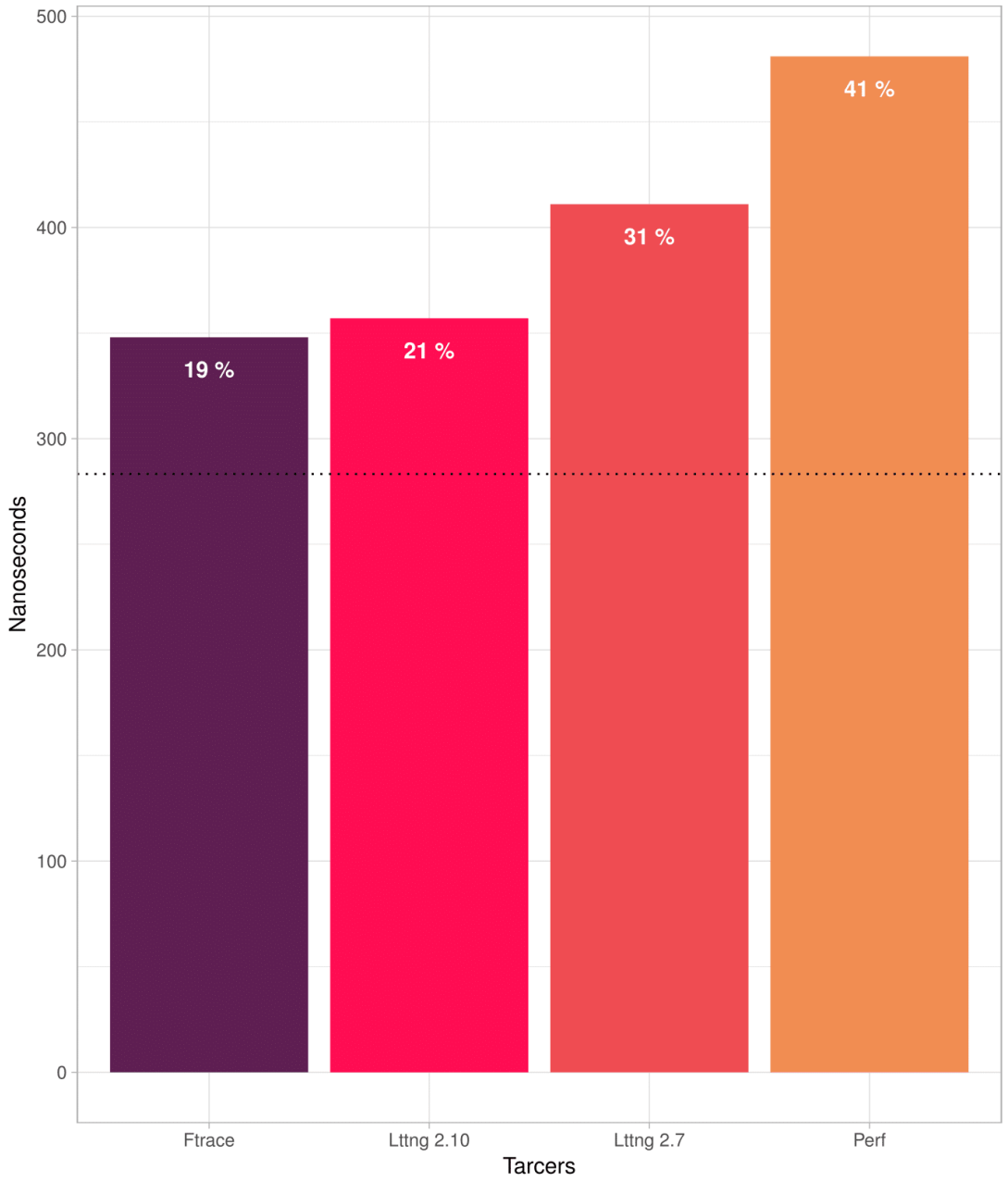
Hypercall Channel

Hypercall in a nutshell



Which tracer to use ?

Host Tracers Overhead (hypercall)



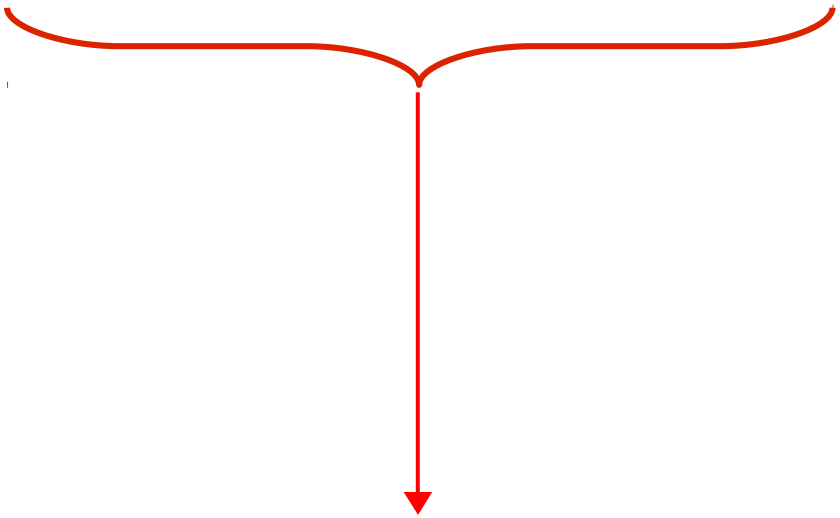
Event Aggregation

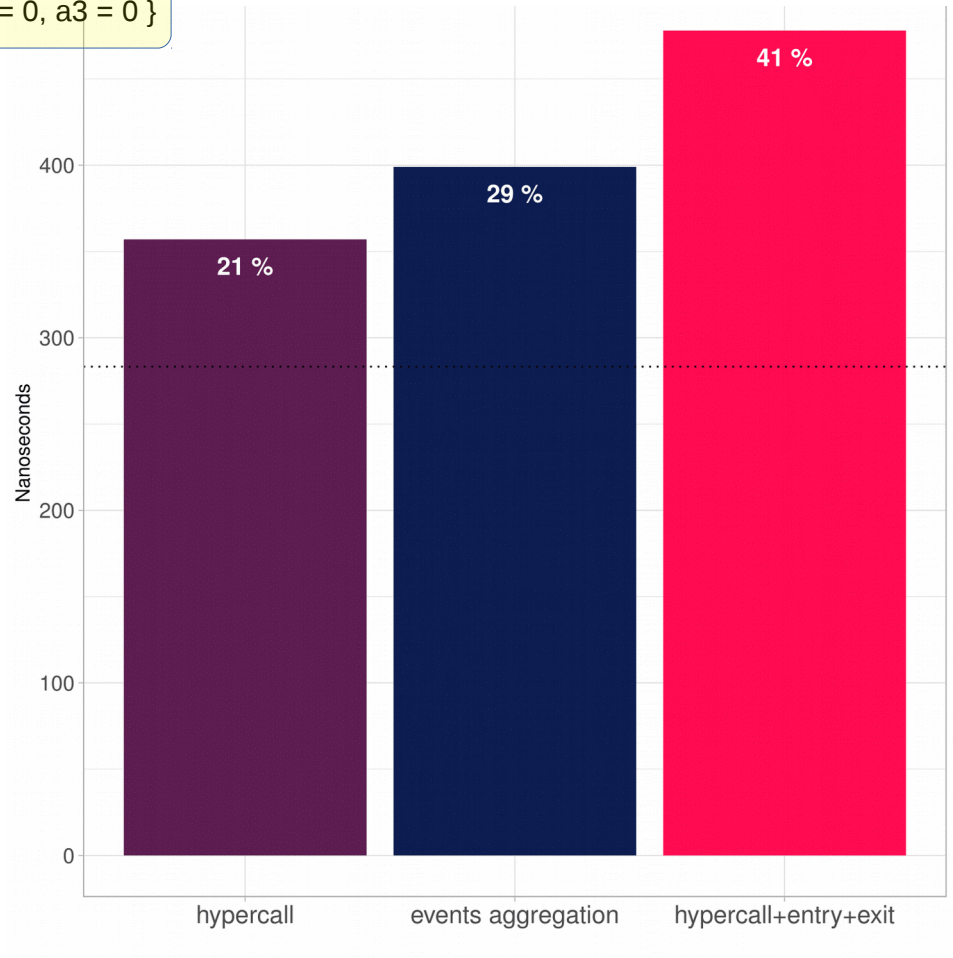
Tracing hypercalls on host using Lttng

kvm_x86_exit: { cpu_id = 0 }, { exit_reason = 18, guest_rip = **94139651574911**, isa = 1... }

kvm_x86_hypercall: { cpu_id = 0 }, { nr = **1000**, a0=**0xffffffff89c2ce60**, a1 = 0, a2 = 0, a3 = 0 }

kvm_x86_entry: { cpu_id = 0 }, { vcpu_id = **2** }

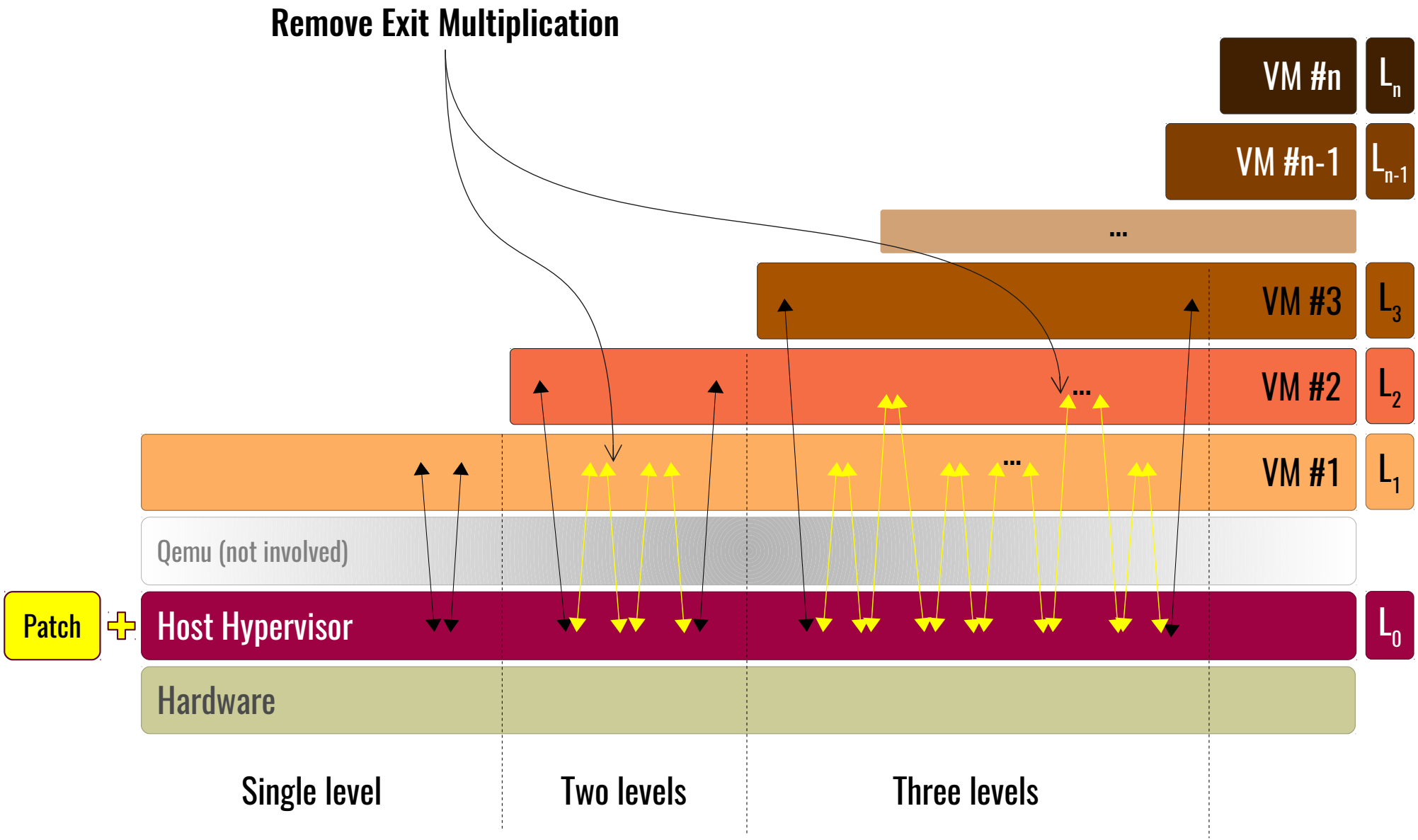




hypergraph_host: { nr = **1000**, a0 = **0xffffffff89c2ce60**, a1 = 0, a2 = 0, a3 =0, vcpu_id = **2**, guest_rip = **94139651574911**, **exit_overhead** = 85 }

Nested VMs

Nested Virtualization



Micro-Benchmarks

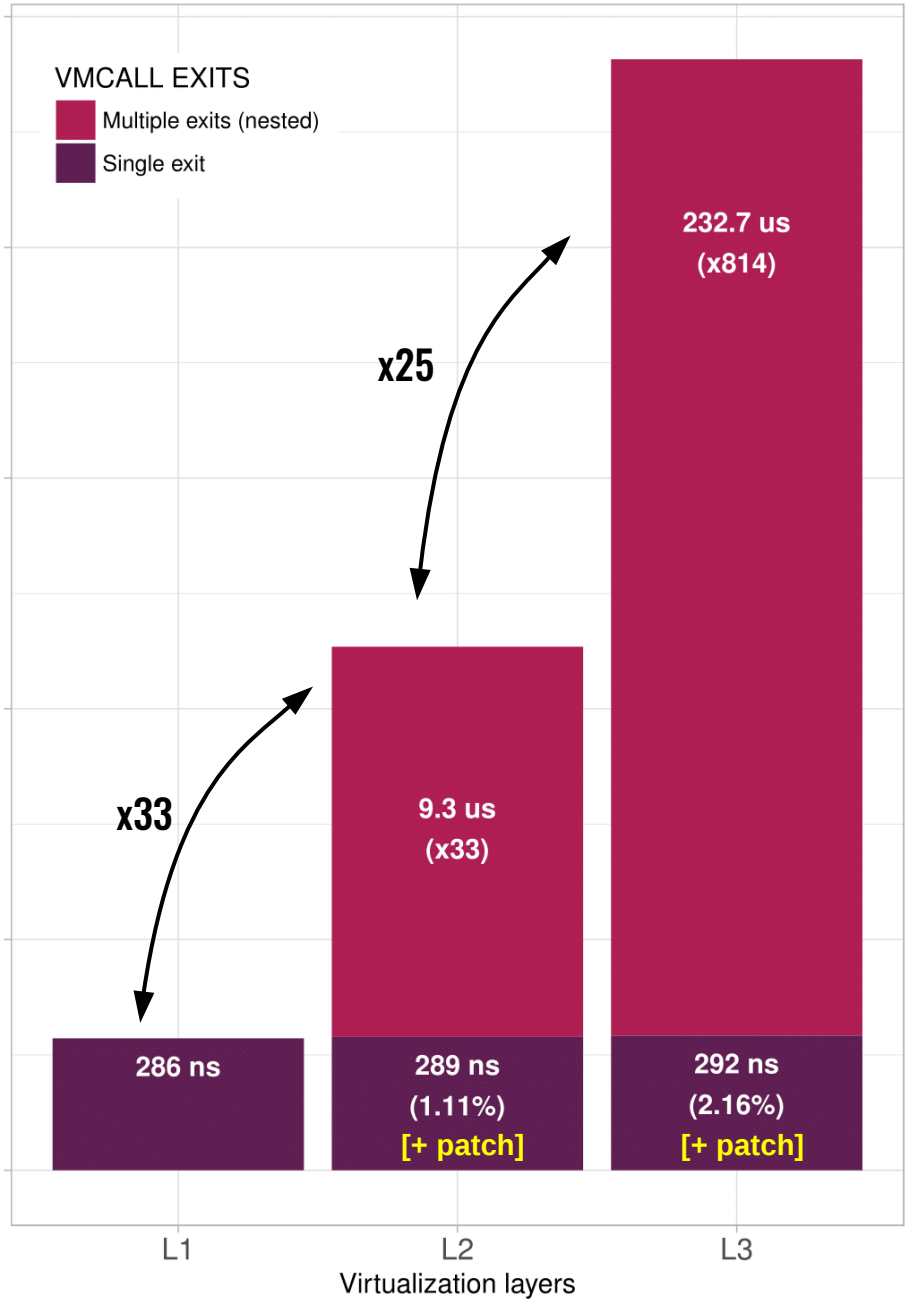
Hypercall Overhead

Hardware : Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz (x86_64)

EPT-on-EPT-on-EPT

Exit-handling code in the hypervisor is slower when run in L1 or L2 than the same code running in L0

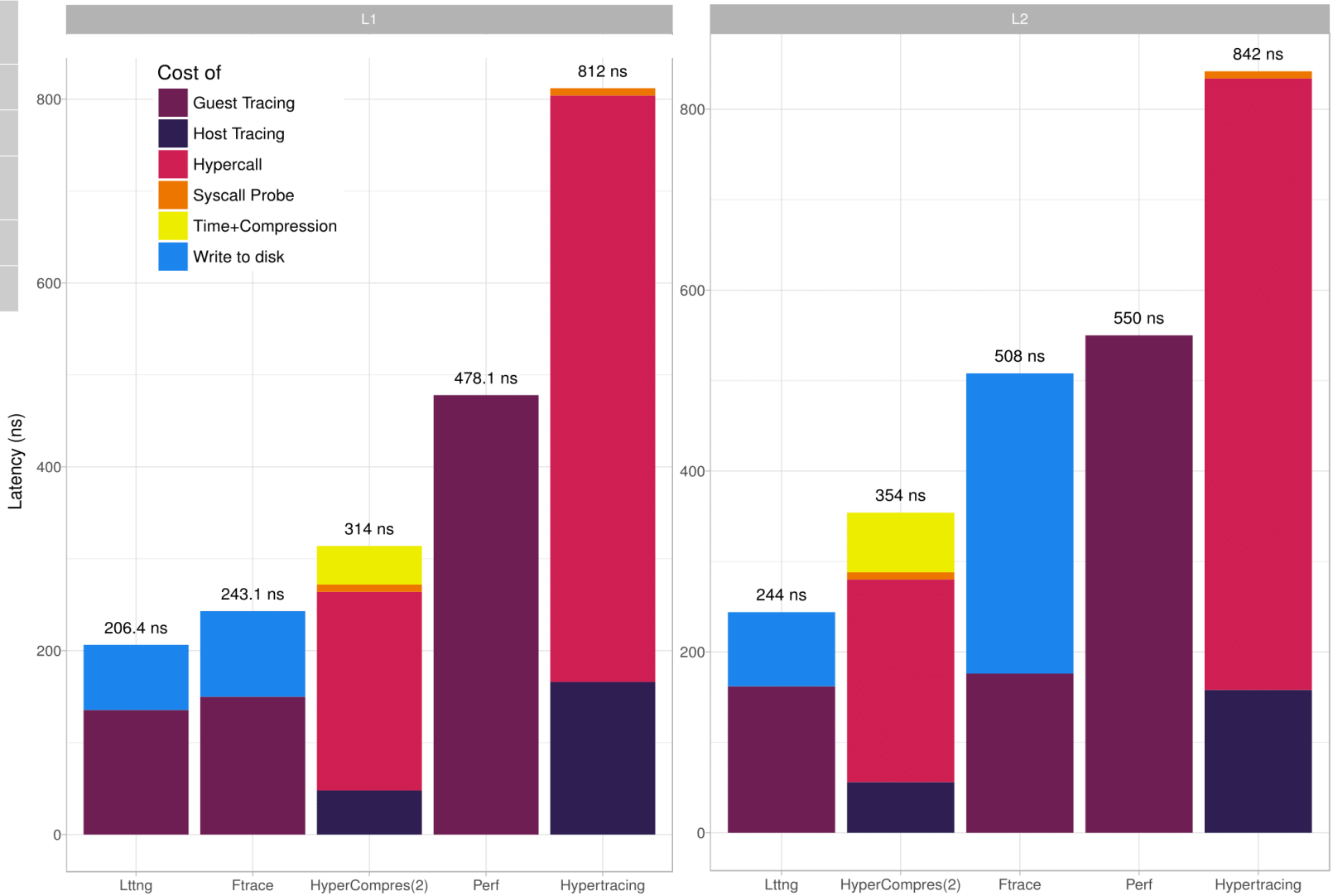
Transition between L1, L2, ... Ln involve an exit to L0 and then an entry.



Micro-Benchmarks (worst-case)

System call

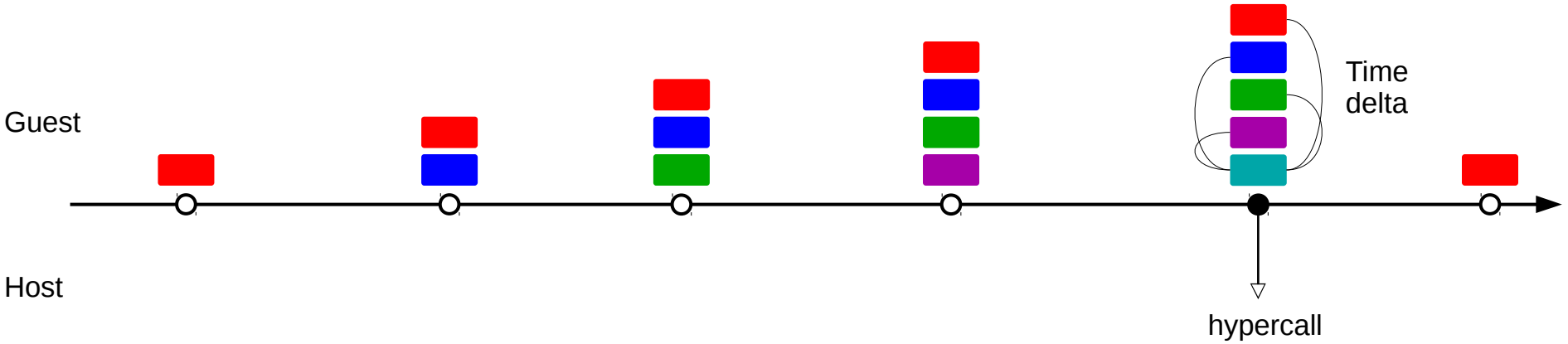
	Nested overhead
Lttng	18 %
Ftrace	x2
Hypertracing + compress (2)	12.6 %
Perf	15 %
Hypertracing	3.6 %



Event Compression

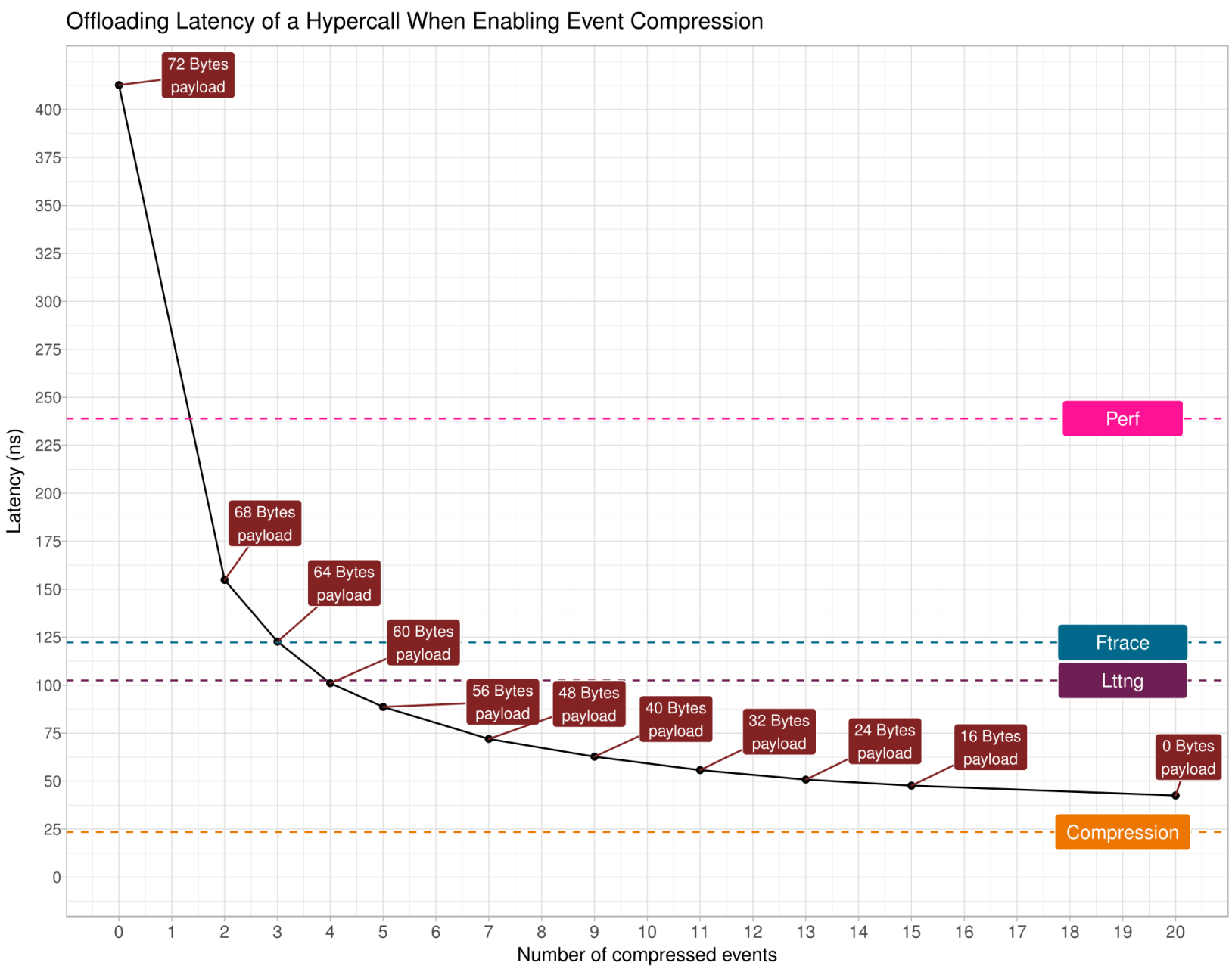
sched_switch example

sched_switch { prev_prio = 0, prev_tid = 0, **next_tid = 10**, prev_state = 0, next_prio = 0, prev_comm = "swapper/1", **next_comm = "migration/1"** }
...
sched_switch { prev_prio = 0, **prev_tid = 10**, **next_tid = 11**, prev_state = 1, next_prio = 0, **prev_comm = "migration/1"**, **next_comm = "ksoftirqd/1"** }
...
sched_switch { prev_prio = 0, **prev_tid = 11**, **next_tid = 12**, prev_state = 1, next_prio = 0, **prev_comm = "ksoftirqd/1"**, **next_comm = "kworker/1:0"** }
...
sched_switch { prev_prio = 0, **prev_tid = 12**, **next_tid = 0**, prev_state = 1, next_prio = 0, **prev_comm = "kworker/1:0"**, **next_comm = "swapper/1"** }
...
sched_switch { prev_prio = 0, **prev_tid = 0**, **next_tid = 10**, prev_state = 0, next_prio = 0, **prev_comm = "swapper/1"**, **next_comm = "migration/1"** }



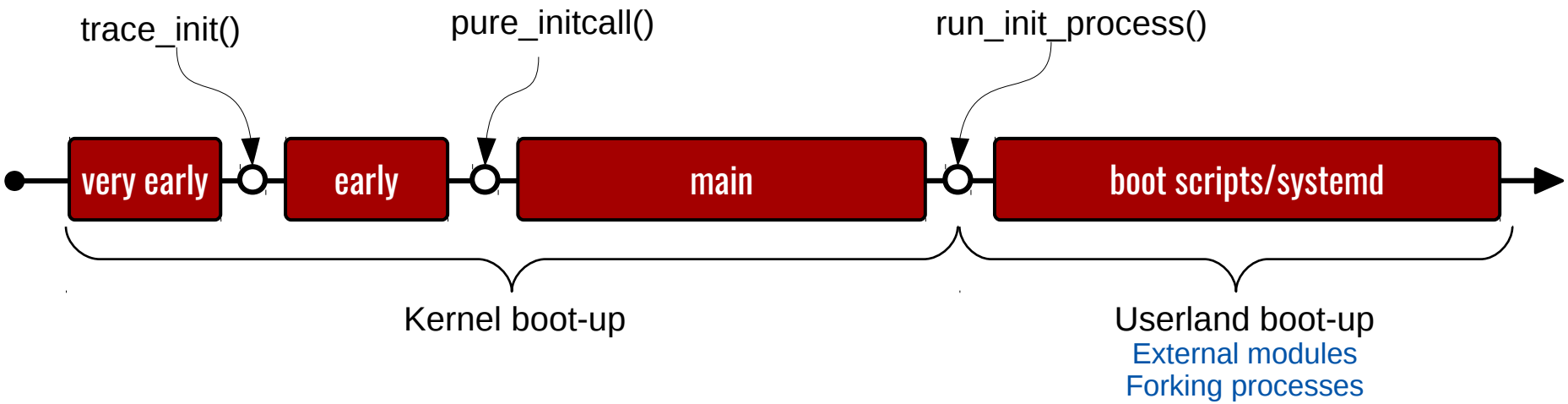
prio -> 8 bits
state -> 8 bits
pid_max -> 15 bits
time_delta -> 32 bits

Event Compression

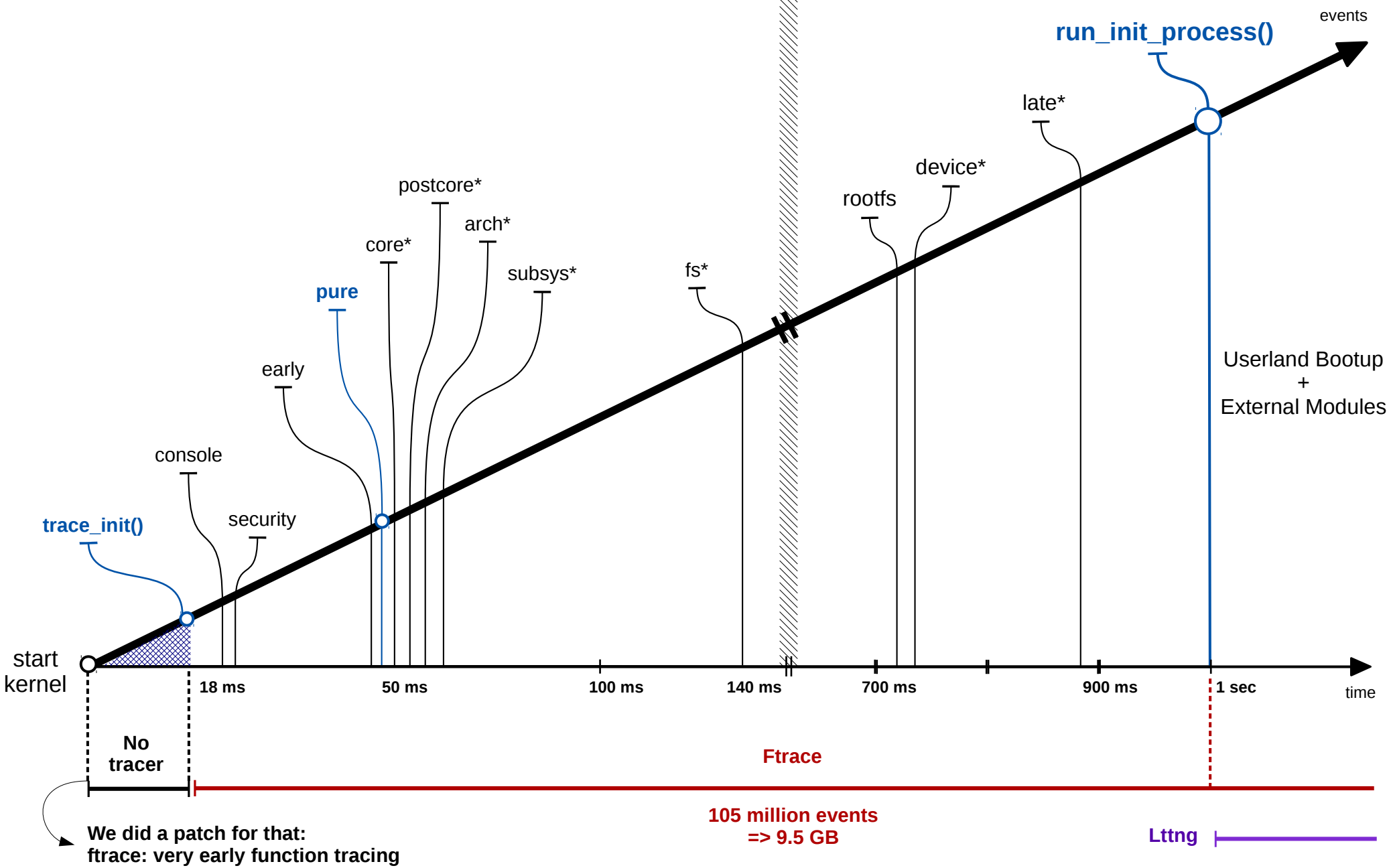


Boot-up Analysis

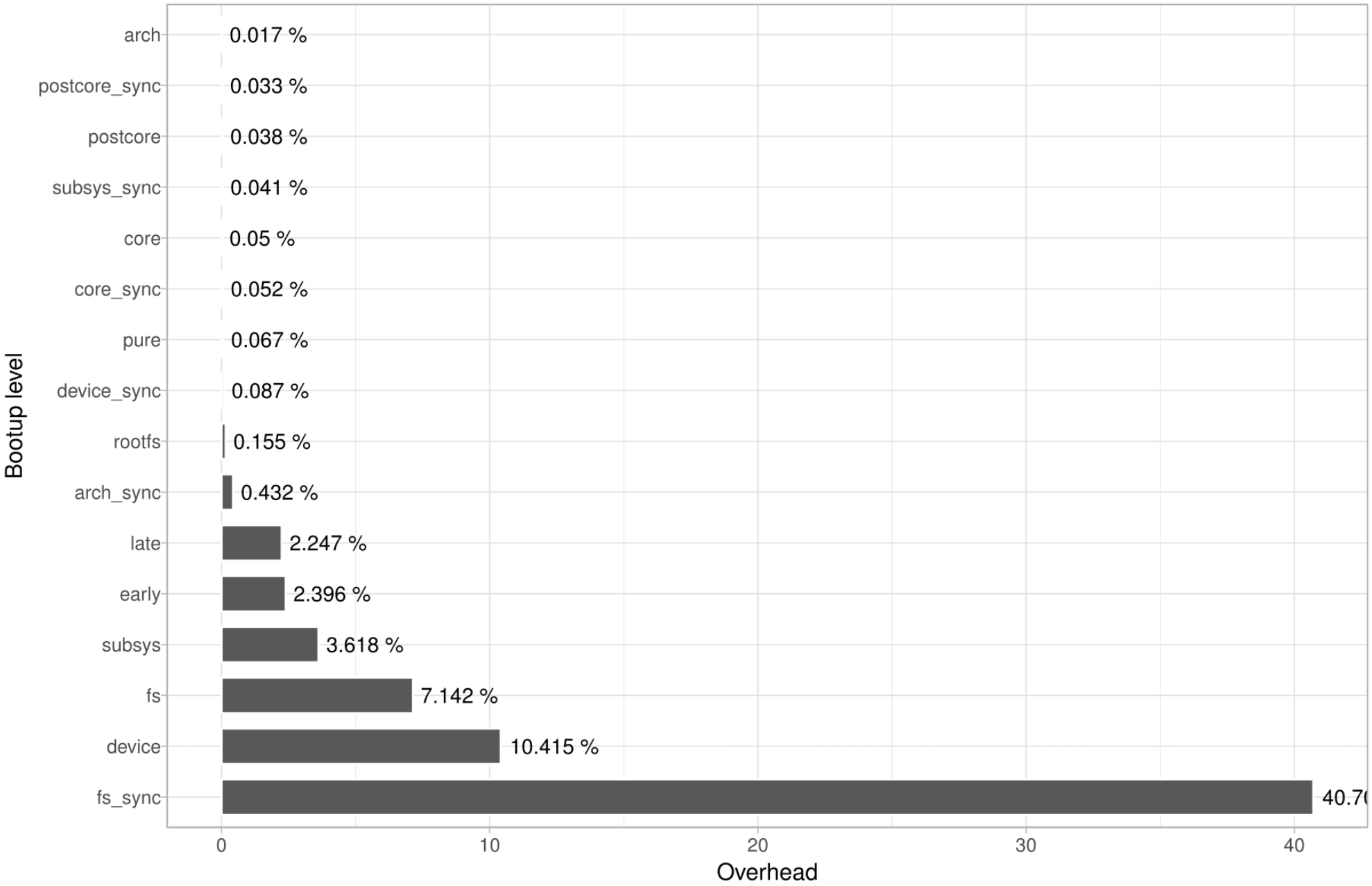
Boot-up phases



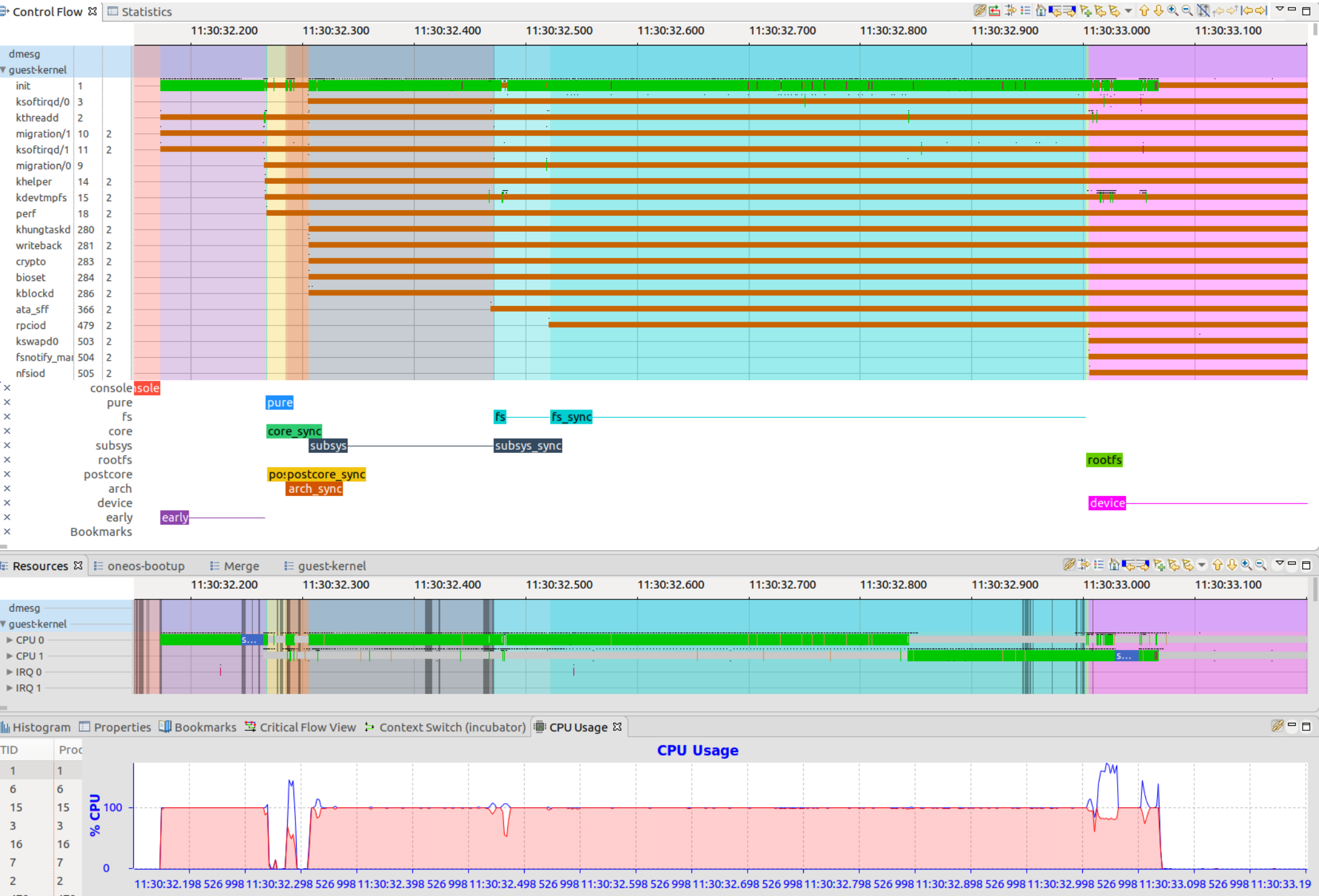
Boot-up Sequence



Boot-up level tracing overhead

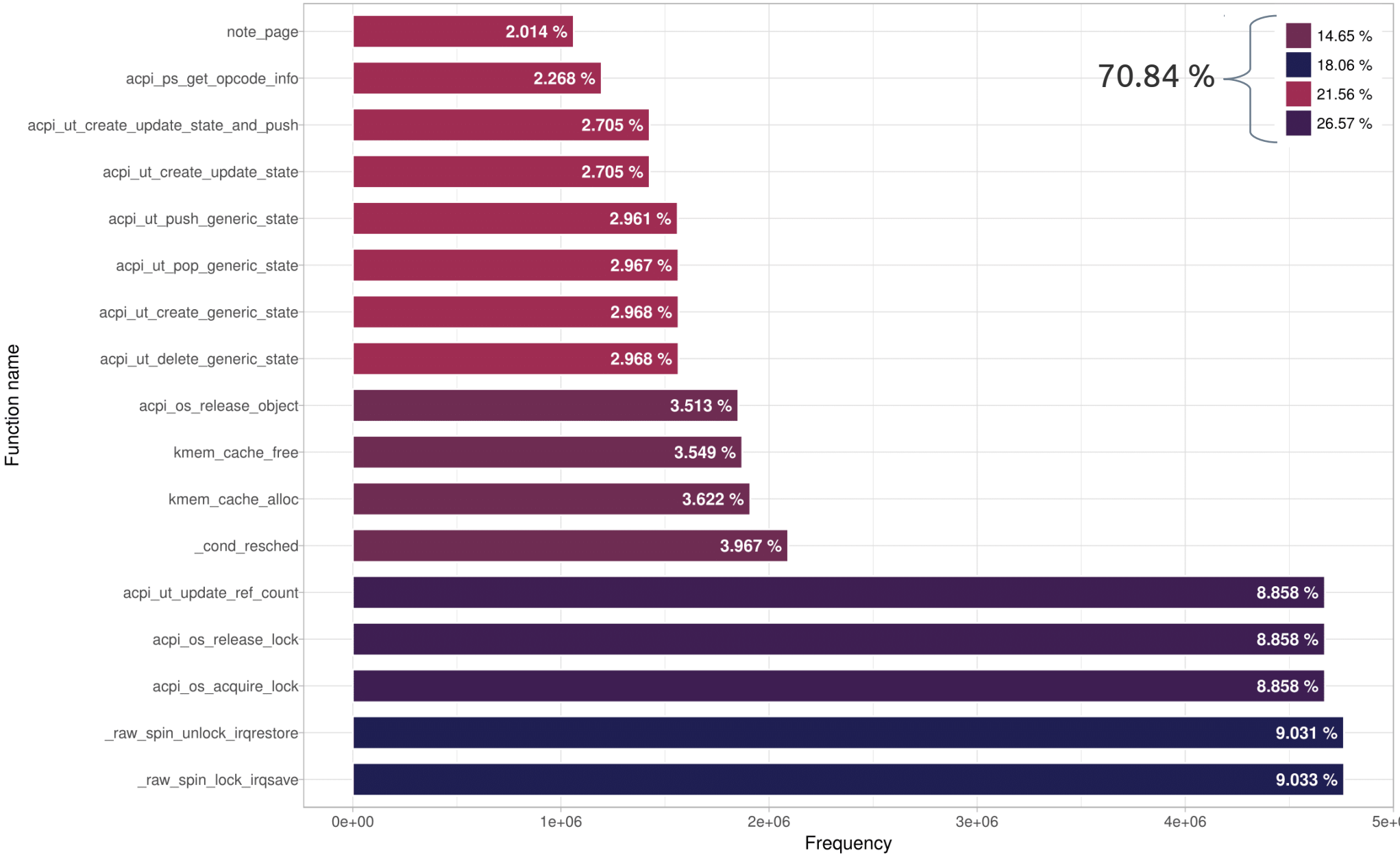


Boot levels : Marker event

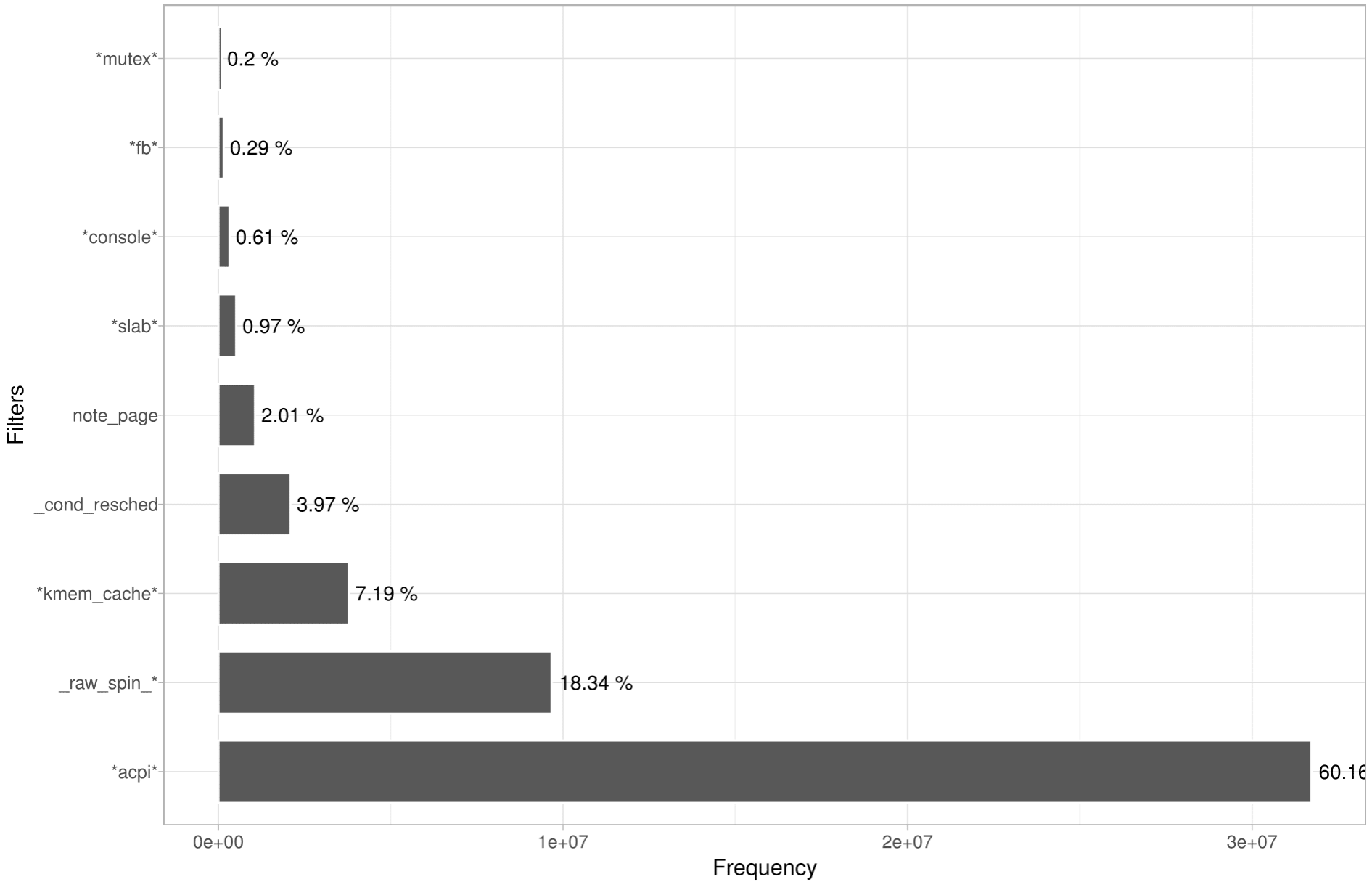


Dynamic Analysis

Top 17 most called functions



Dynamic Analysis



Boot-up Tracing

Function tracing

Host Tracing	Guest			Events	Trace Size	Time
	Tracer	Optimization & Configuration				
-	**Baseline**	-		-	-	868.52 ms
	Function graph	Function entries & exits*		168 M	Buffer size	22 secs
	Hypergraph	Function entries & exits		-	-	1min 08s
Lttng	Hypergraph	Function entries & exits		322 M	9.4 GB	1min 20s
		Event Aggregation (**Host opt.**)		105 M	6.7 GB	57 secs
		Event Compression (**Guest opt.**)		163 M	4.8 GB	40 secs
		Aggr + Comp		52 M	3.4 GB	30 secs
		Aggr + Comp + Filtering (**Guest opt.**)		**3.2 M**	**207 MB**	**2.6 secs**
	Hypergraph + Bootlevel	Aggr + Comp + Filtering + Boot level (Guest opt.)	early	20 K	1.4 MB	93 ms
			pure	5 K	0.3 MB	2.6 ms
			core	8 K	0.5 MB	4 ms
			postcore	5 K	0.4 MB	2.7 ms
			arch	32 K	2.1 MB	17 ms
			subsys	115 K	7.5 MB	143 ms
			fs	2.5 M	155 MB	1871 ms
			rootfs	10 K	0.7 MB	6 ms
			device	412 K	26 MB	410 ms
			late	258 K	16 MB	87 ms

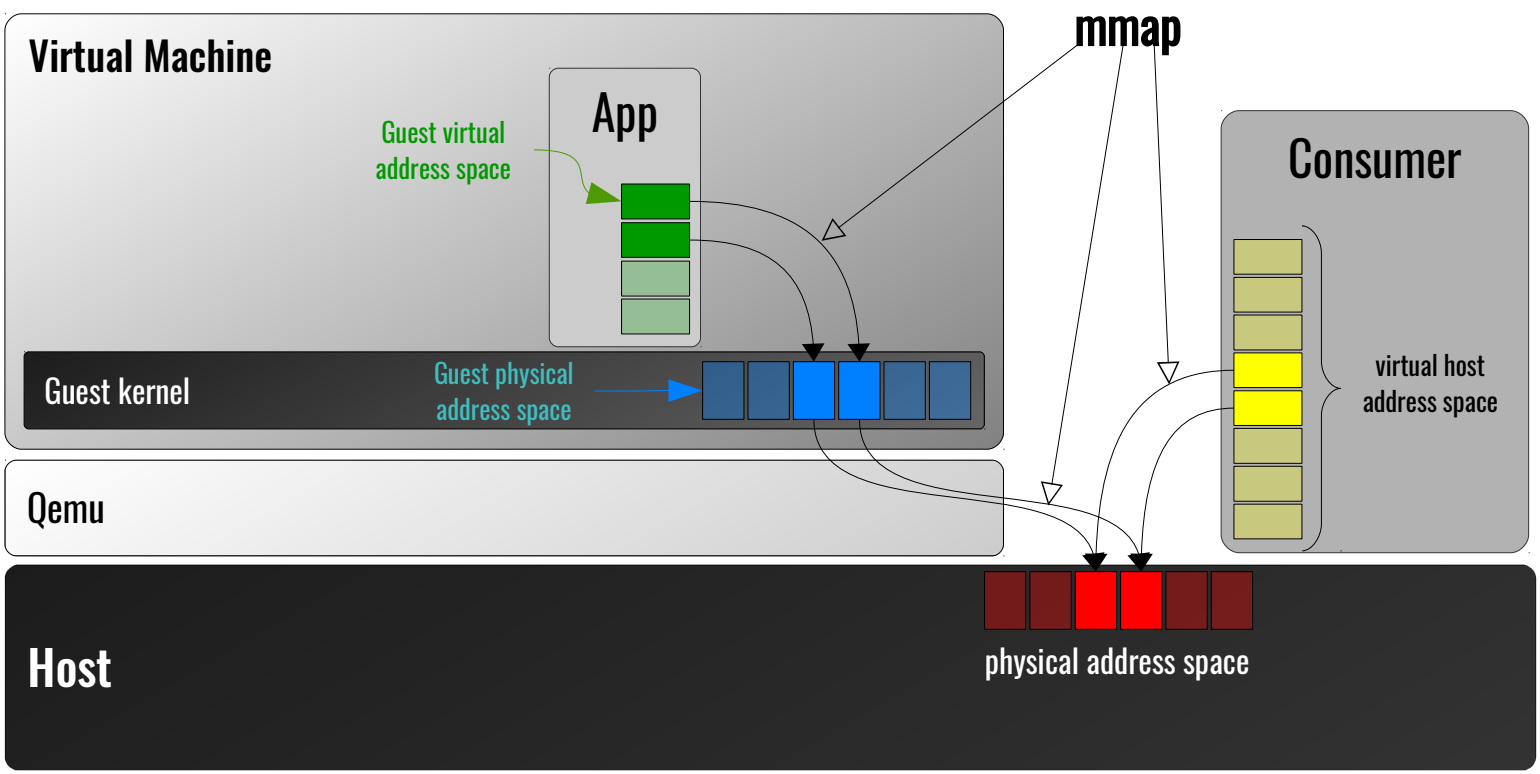
66% overhead

Friendly advice :
Always use function tracing with filters.
Enable it for specific cases

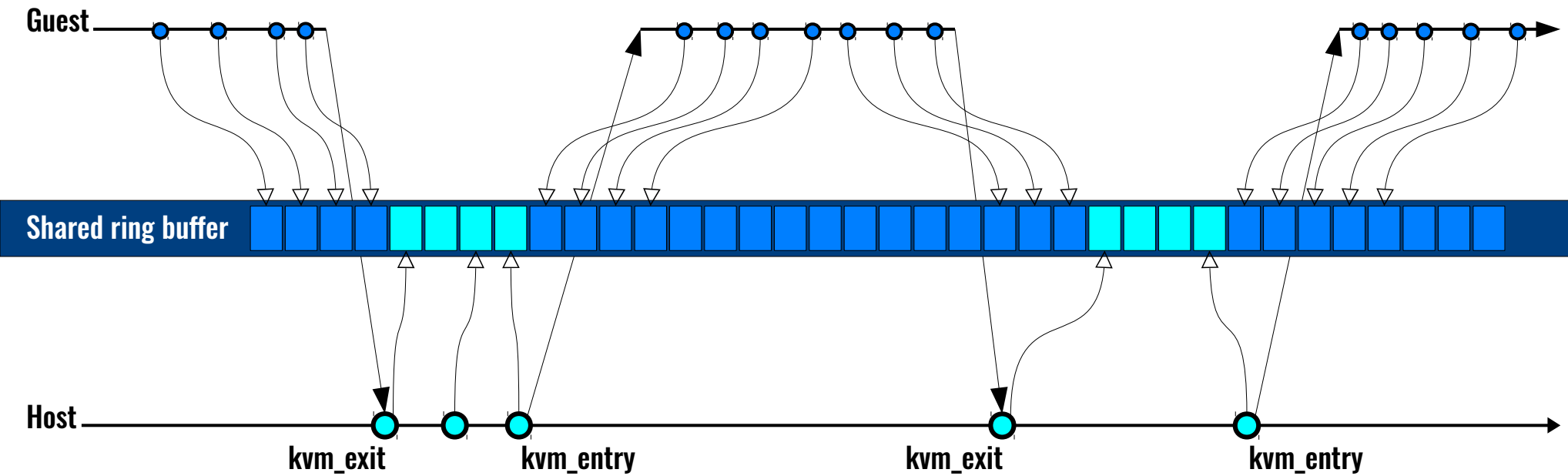
Shared Memory Channel

Zero Copy Shared Memory

Zero copy using mmap



Virtualization awareness

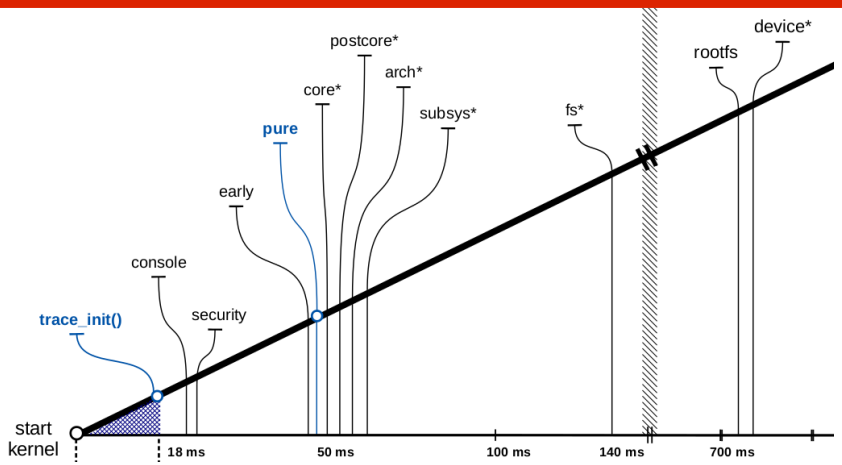


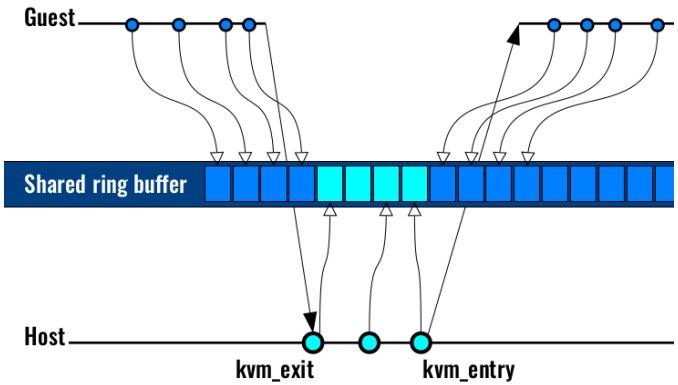
Synchronization ?

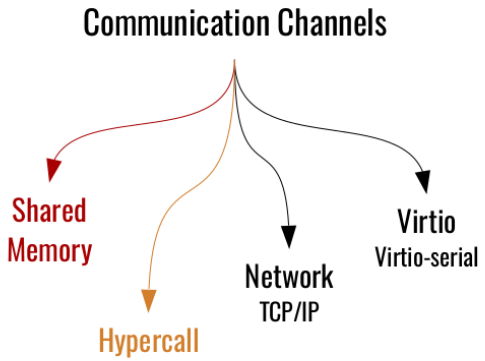
Work in progress ...

Future Work

Compare Shared M. vs Hypercall vs Virtio
Shutdown analysis
Virtualization awareness







Feedbacks & Questions

abderrahmane.benbachir@polymtl.ca

https://github.com/abenbachir

References

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

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

Trampoline: https://www.kernel.org/doc/ols/2009/ols2009-pages-47-54.pdf

Callstack xml analysis: https://gist.github.com/abenbachir/e813790f183945b6f74dc74ecee57c75