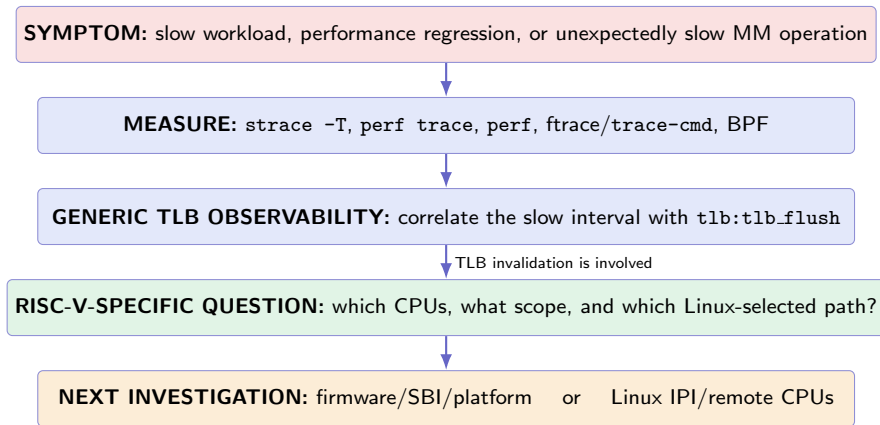


Local, SBI, or IPI? Tracing RISC-V TLB Flush Decisions in Linux

Roman “Hedin” Storozhenko <romeusmeister@gmail.com>

Linux Plumbers Conference 2026

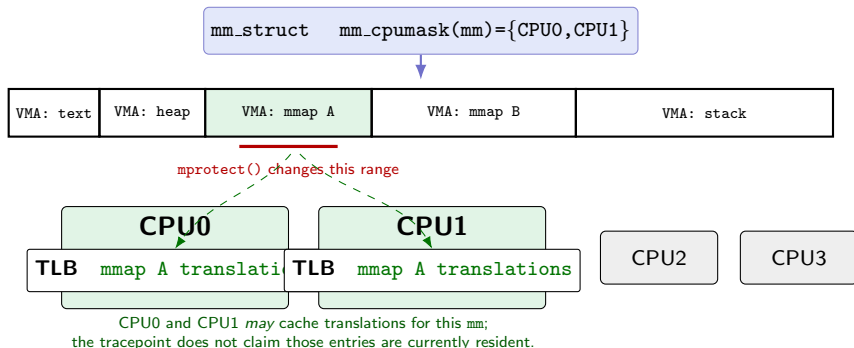
From a Slow Workload to a RISC-V TLB Question



MM activity that may require invalidation

`mprotect()` `munmap()` `mremap()` reclaim migration NUMA balancing

One Address Space, Several VMAs, One CPU Footprint



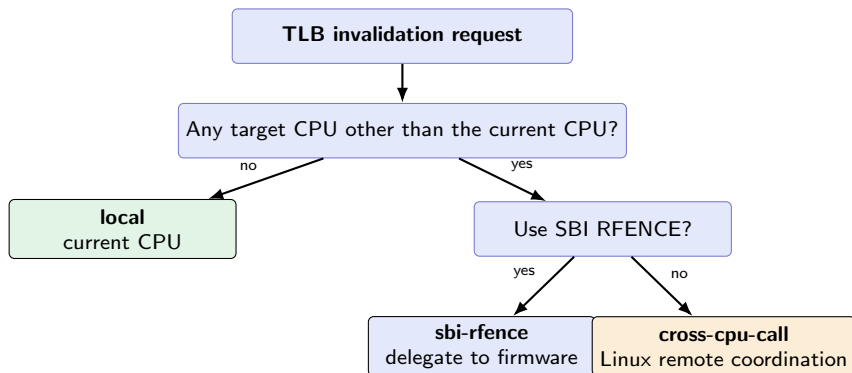
Local footprint

`mm_cpumask={0}`
no other target CPU

Remote footprint

`mm_cpumask={0,1}`
CPU1 may need invalidation too

How RISC-V Linux Chooses the Flush Path



Flush path = how Linux chooses to carry out the TLB invalidation.

Generic TLB Event First; RISC-V Event Next

Generic: `tlb:tlb_flush`

```
tlb_flush:  
  pages=64  
  reason=remote shutdown
```

- ▶ confirms TLB flush activity
- ▶ gives generic flush reason and page count
- ▶ useful for correlation with the slow interval

RISC-V:

`riscv_tlb:riscv_tlb_flush_path`

```
riscv_tlb_flush_path:  
  size=0x40000 stride=0x1000  
  target_cpus=0-1 weight=2  
  scope=range  
  path=sbi-rfence
```

- ▶ request context
- ▶ exact target CPUs
- ▶ requested scope
- ▶ Linux-selected RISC-V path

The new event does not replace the generic one.

Generic tracing says a flush happened; the RISC-V event explains how Linux chose to carry it out.

How to Read riscv_tlb_flush_path

```
riscv_tlb_flush_path: start=0x7fffb4bc2000 size=0x40000 stride=0x1000  
  asid=0x12b has_mm=1 target_mask_weight=2 target_cpus=0-1  
  scope=range path=cross-cpu-call
```

Request

Field	Meaning	Example
<code>start</code>	first virtual address	<code>0x7fff...</code>
<code>size</code>	requested range size	<code>0x40000</code>
<code>stride</code>	invalidation step / mapping size	<code>0x1000</code>
<code>asid</code>	hardware-visible address-space ID	<code>0x12b</code>
<code>has_mm</code>	associated with a specific mm?	<code>1</code>

Targets and decision

Field	Meaning	Example
<code>target_cpus</code>	which CPUs	<code>0-1</code>
<code>target_mask_weight</code>	how many CPUs	<code>2</code>
<code>scope</code>	requested extent	<code>range</code>
<code>path</code>	selected flush mechanism	<code>cross-cpu-call</code>

Scope values

Value	Meaning	Value	Meaning
<code>single</code>	request no larger than one stride	<code>range</code>	finite request larger than one stride
<code>address-space</code>	complete address space for a specific mm	<code>all</code>	global request, no specific mm

Demo: Same MM Operation, Different CPU Footprint

Measured operation: 64-page mprotect()

Case A: local footprint

```
mm_cpumask={0}  
  
target_mask_weight=1  
target_cpus=0  
scope=range  
path=local
```

Case B: remote footprint

```
mm_cpumask={0,1}  
  
target_mask_weight=2  
target_cpus=0-1  
scope=range  
path=sbi-rfence  
sbi_call ... fid=2
```

Same kind of MM request, different CPU footprint, different Linux-selected path.

► Recorded terminal demo on asciinema

What Did Testing Actually Exercise?

Controlled scenarios

QEMU configuration	Workload	Scope	Path
virt + SBI RFENCE	local 64-page mprotect()	range	local
virt + SBI RFENCE	remote 64-page mprotect()	range	sbi-rfence
virt + APLIC+IMSIC	remote 64-page mprotect()	range	cross-cpu-call
virt + APLIC+IMSIC	MADV_PAGEOUT reclaim	all	cross-cpu-call

Path coverage

- ▶ all three path values were **explicitly exercised**
- ▶ local
- ▶ sbi-rfence
- ▶ cross-cpu-call

Scope coverage

- ▶ explicitly targeted: **range, all**
- ▶ also observed in complete traces: **single, address-space**

The table shows the controlled experiments; incidental process/MM activity supplied the additional single and address-space observations.

Takeaways and Upstream Status

Takeaways

1. A slow workload/MM operation is first correlated with MM and TLB activity using existing tools.
2. The generic TLB event says a flush happened; the RISC-V event adds **request + targets + scope + Linux-selected flush path**.
3. The CPU mask tells **which** CPUs; its weight tells only **how many**.

Upstream status

v1 $\implies$ **Steven Rostedt review** $\implies$ v2 (**CI PASS**) $\implies$ **v2 RESEND (CI PASS)**

v2 uses `trace_call_<event>()` after the explicit enabled check, following Steven's review; the unchanged v2 RESEND also passed CI.

► Latest v2 RESEND

► Recorded demo

► Workloads

Tracepoint contract: record the request, target CPUs, scope, and Linux-selected flush path **before dispatch**; do not claim completion, latency, hardware TLB contents, or firmware-internal RFENCE behavior.

Backup: Reproduction, Latest Patch, and Demo

Validation workloads and QEMU reproduction instructions

- ▶ https://github.com/Romeus/tlb_workloads/

Latest patch posting: v2 RESEND

- ▶ https://lore.kernel.org/lkml/20260924-tlb_tracepoint-v2-1-4e3e78ef5cdb@gmail.com/

Recorded terminal presentation

- ▶ <https://asciinema.org/a/1266683>

LPC contribution

- ▶ <https://lpc.events/event/20/contributions/2342/>

Backup: Review History and CI

Patch discussion and submission history

- ▶ **v1** + Steven Rostedt review
▶ v1 thread + review
- ▶ **original v2** – 30 Aug 2026
▶ original v2
- ▶ **v2 RESEND** – 24 Sep 2026
▶ v2 RESEND

RISC-V Patchwork CI

- ▶ original v2
PASS
- ▶ v2 RESEND
PASS (latest)
▶ Latest Patchwork entry

The resend is unchanged; it is the latest public posting and has its own newer CI result.

Backup: Path Selection in `__flush_tlb_range()`

```
if (cpumask_any_but(cmask, cpu) >= nr_cpu_ids) {
    riscv_tlb_trace_flush_path(..., PATH_LOCAL);
    local_flush_tlb_range_asid(...);
} else if (riscv_use_sbi_for_rfence()) {
    riscv_tlb_trace_flush_path(..., PATH_SBI_RFENCE);
    sbi_remote_sfence_vma_asid(...);
} else {
    riscv_tlb_trace_flush_path(...,
        PATH_CROSS_CPU_CALL);
    on_each_cpu_mask(...);
}
```

Backup: Steven Rostedt's v1 → v2 Suggestion

v1 helper

```
if (!trace_riscv_tlb_flush_path_enabled())  
    return;  
  
scope = riscv_tlb_get_flush_scope(...);  
trace_riscv_tlb_flush_path(...);
```

v2 helper

```
if (!trace_riscv_tlb_flush_path_enabled())  
    return;  
  
scope = riscv_tlb_get_flush_scope(...);  
trace_call__riscv_tlb_flush_path(...);
```

Steven Rostedt suggested the direct `trace_call__<event>()` form because the helper already performed the explicit enabled check, avoiding a second tracepoint static-key test.

Backup: Scope Semantics

scope	Meaning
single	request no larger than one stride
range	finite request larger than one stride
address-space	FLUSH_TLB_MAX_SIZE with a specific mm
all	FLUSH_TLB_MAX_SIZE without a specific mm

scope describes the request presented to the RISC-V implementation. It deliberately does not expose the later local-helper implementation choice.

Backup: CPU Mask vs Weight

- ▶ `target_cpus`: **which** CPUs Linux intends to cover.
- ▶ `target_mask_weight`: **how many** CPUs are in that mask.

`target_cpus=0,2-3` `target_mask_weight=3`

`weight=3`

cannot distinguish

`cpus=0-2` from `cpus=0,7,12`

CPU identity is needed to correlate the request with scheduler, IPI, and firmware activity.

Backup: The 64-Entry Local Threshold

This is not path selection

The threshold is used later when Linux executes its local range helper on a CPU.

$$\begin{array}{ccc} \text{entries} = \text{ceil}(\text{size} / \text{stride}) & & \\ \downarrow & & \\ \text{entries} \leq 64 & \Rightarrow & \text{per-address invalidation} \\ \text{entries} > 64 & \Rightarrow & \text{full-ASID local flush} \end{array}$$

Why 64 and 65 pages?

64 pages $\rightarrow$ scope=range 65 pages $\rightarrow$ scope=range

Both remain finite range *requests*; the threshold only changes the later local execution strategy.

Backup: The Controlled Workload

Local mm footprint

```
taskset -c 0 ./tlb_workload-riscv64 local 64 mprotect 0
```

Remote mm footprint

```
taskset -c 0 ./tlb_workload-riscv64 remote 64 mprotect 0 1
```

The worker in the remote case uses the same `mm` on CPU1 before CPU0 performs the measured operation. This creates a controlled multi-CPU target footprint.

Backup: Reaching cross-cpu-call

QEMU machine

```
-M virt,aia=aplic-imsic
```

Observed remote 64-page request

```
size=0x40000 stride=0x1000 has_mm=1  
target_mask_weight=2 target_cpus=0-1  
scope=range path=cross-cpu-call
```

No corresponding SBI RFENCE `sbi_call` appears in the measured `mprotect()` interval.

Backup: Reaching scope=all

Userspace trigger

```
./tlb_scope_all-riscv64 <writable-mmap-capable-file>
```

Representative event

```
start=0 size=0xffffffffffffffff stride=0x1000  
asid=0xffffffffffffffff has_mm=0  
target_mask_weight=3 target_cpus=0,2-3  
scope=all path=cross-cpu-call
```

Backup: Validation Coverage

Runtime

- ▶ local – PASS
- ▶ SBI RFENCE – PASS
- ▶ cross-CPU call – PASS
- ▶ all four scope values observed

RISC-V Patchwork CI

- ▶ original v2 – **PASS**
- ▶ v2 RESEND – **PASS** (latest)
- ▶ RV32 defconfig – PASS
- ▶ RV64 GCC/Clang
allmodconfig – PASS
- ▶ NOMMU builds – PASS

Backup: TLB Shutdown Performance and Debugging

Practical debugging / performance references

- ▶ Erik Rigtorp, *Latency Implications of Virtual Memory*
Demonstrates VM-induced latency, inspects TLB shutdowns, and uses `perf stat -e tlb:tlb_flush`.
<https://rigtorp.se/virtual-memory/>
- ▶ Erik Rigtorp, *Low Latency Tuning Guide*
Includes `tlb:tlb_flush` and `/proc/interrupts` as tools for identifying shutdown activity.
<https://rigtorp.se/low-latency-guide/>
- ▶ JabPerf, *TLB Shutdowns: How To Deter or Disarm Them*
Case study using shutdown counters and tracing to identify the source of remote invalidations.
<https://www.jabperf.com/how-to-deter-or-disarm-tlb-shutdowns/>
- ▶ Amit et al., *Don't shoot down TLB shutdowns!*, EuroSys 2020
Academic motivation for reducing TLB shutdown overhead.
<https://doi.org/10.1145/3342195.3387518>