



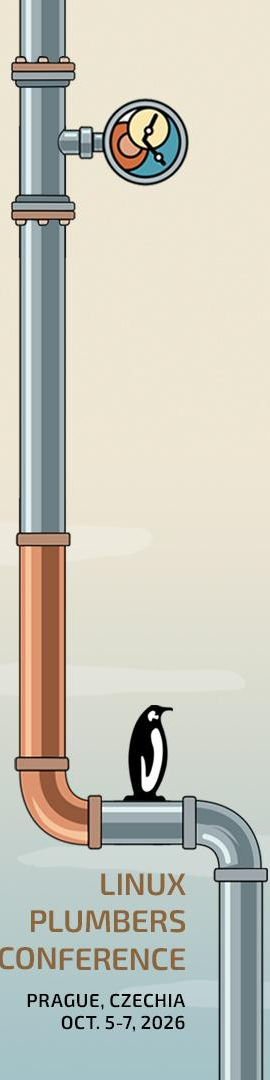
LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Bridging the VM Boundary

Scheduling Passthrough via pvsched and sched_ext

Vineeth Pillai, Josh Don



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



Background



ChromeOS

Supports Android apps running inside an Android VM. Interactive workloads (audio/video) need high-priority, glitch-free task scheduling.

Making the host kernel aware of VM scheduling requirements significantly reduces latencies and improves overall user experience.



Shared Tenant VMs

VMs contain a mix of jobs of varying priorities

How to prevent priority inversion and fairly share resources between VMs?



Related Work

IBM Proposal

Guest steers threads away from heavily preempted VCPUs.

Phantom Tracker

Tunes OpenMP in response to overcommit-induced wait time, via IVSHMEM.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



What is pvsched (Paravirt Scheduling)?



WHAT

Cooperative Framework

Host and guest share scheduling-related information to make optimal decisions regarding task priority and placement.



MECHANISM

Shared Memory & Protocol

Shared memory region setup between guest and host for communication.

Defines memory layout, shared info details, and policy rules.



HOW

Handshake & Negotiate

During guest boot-up, host and guest execute a handshake to negotiate protocol and exchange shared memory addresses.

Runtime Scheduling

Once handshake completes, scheduling info is actively exchanged to drive policy-based decisions.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Brief History of pvsched

- V1: <https://lkml.org/lkml/2023/12/13/1789>
 - Kvm does most of the heavy lifting: handshake, policies, scheduling decision
 - Upstream was against having all these logic in kvm
- V2: <https://lwn.net/Articles/968242/>
 - Kvm does the handshake with the guest
 - Policy and scheduling decision designed to be implemented separately as a kernel module or a BPF program
 - module/BPF registers to kvm for receiving callbacks on interested events
 - Kvm maintainers does not like the idea of having the handshake also in kernel
- V3: (preparing to send upstream)
 - Handshake in the VMM
 - Policies and scheduling decisions in a BPF program
 - Could be a kernel module as well
 - Loaded by VMM or a standalone pvsched loader



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



New pvsched(v3) mechanism

The pvsched-v3 architecture relies on several tightly integrated but isolated components to enable cooperative virtualization:

- **pvsched_guest driver:** A guest kernel module that hooks into vital 'sched_switch' and 'irq' tracepoints to directly write scheduling priority and execution context to shared memory.
- **Shared Memory Page:** A dedicated 4KB page allocated per vCPU, segmented into a Header, a Guest Area for writing state, and a Host Area for reading status and hints.
- **Virtual PCI Device:** Emulated by the VMM (such as crosvm or QEMU) to facilitate discovery and setup, exposing a Control MMIO Doorbell via BAR0 and a GPA Table Region via BAR1.
- **Host-Side pvsched Module:** Pins shared guest pages into memory, provides safe boosting helper functions, and strictly enforces anti-starvation timers.
- **eBPF/kernel modul Policy Engine:** A pluggable scheduler component, attaching via BPF struct_ops or tracepoints to make dynamic vCPU priority adjustments based on shared state.



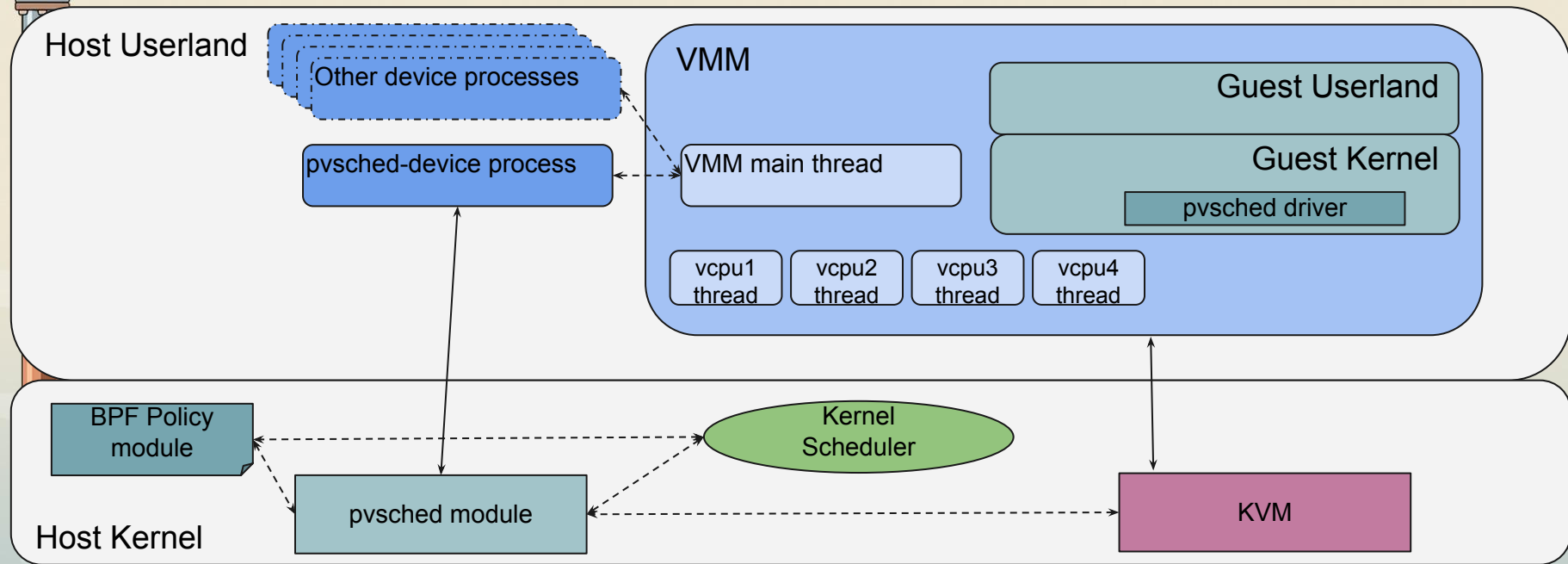
LINUX

PLUMBERS CONFERENCE

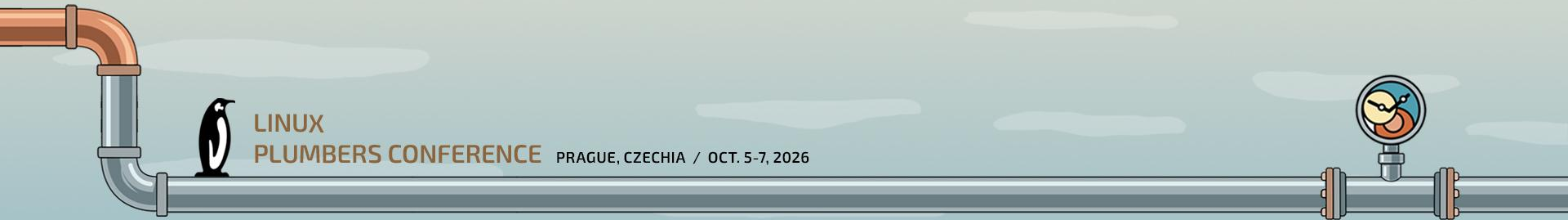
PRAGUE, CZECHIA / OCT. 5-7, 2026



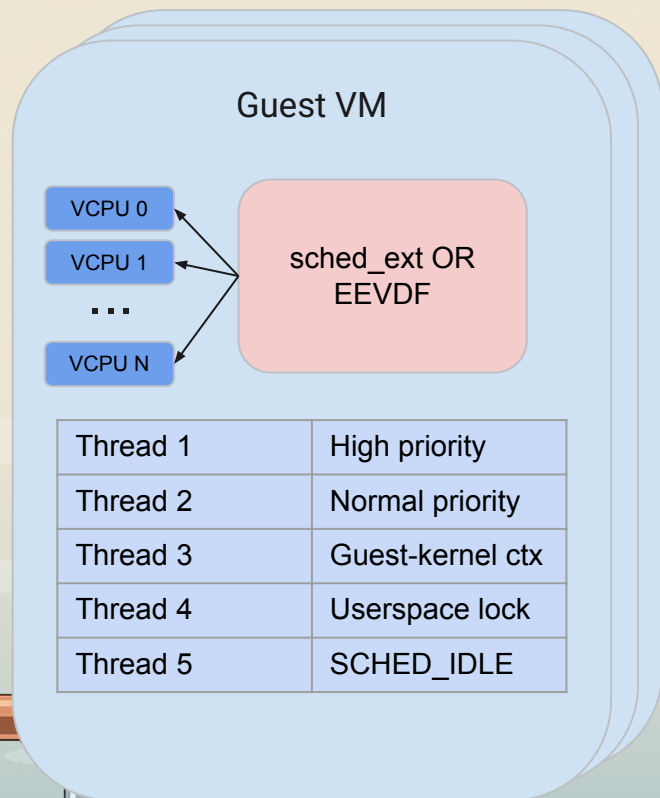
Pvsched V3 Architecture



Paravirt scheduling communication



Paravirt scheduling communication

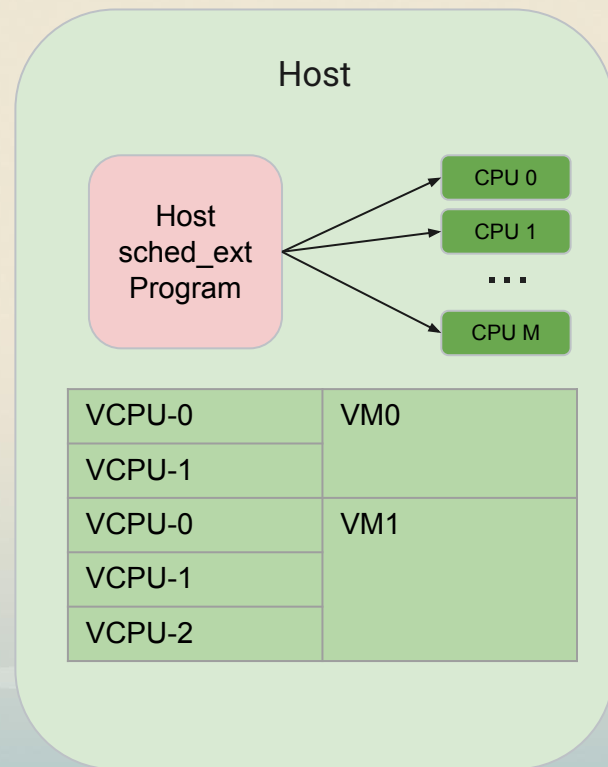
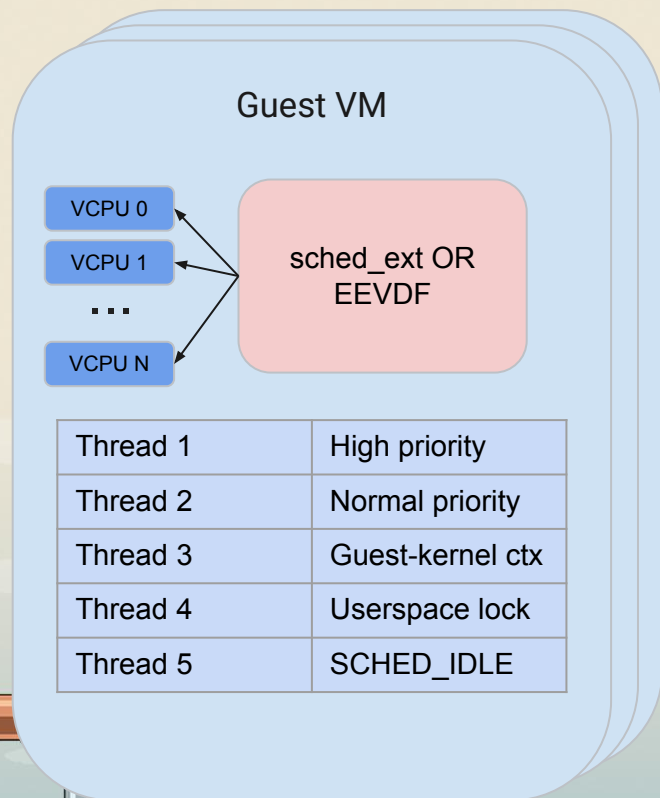


LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Paravirt scheduling communication



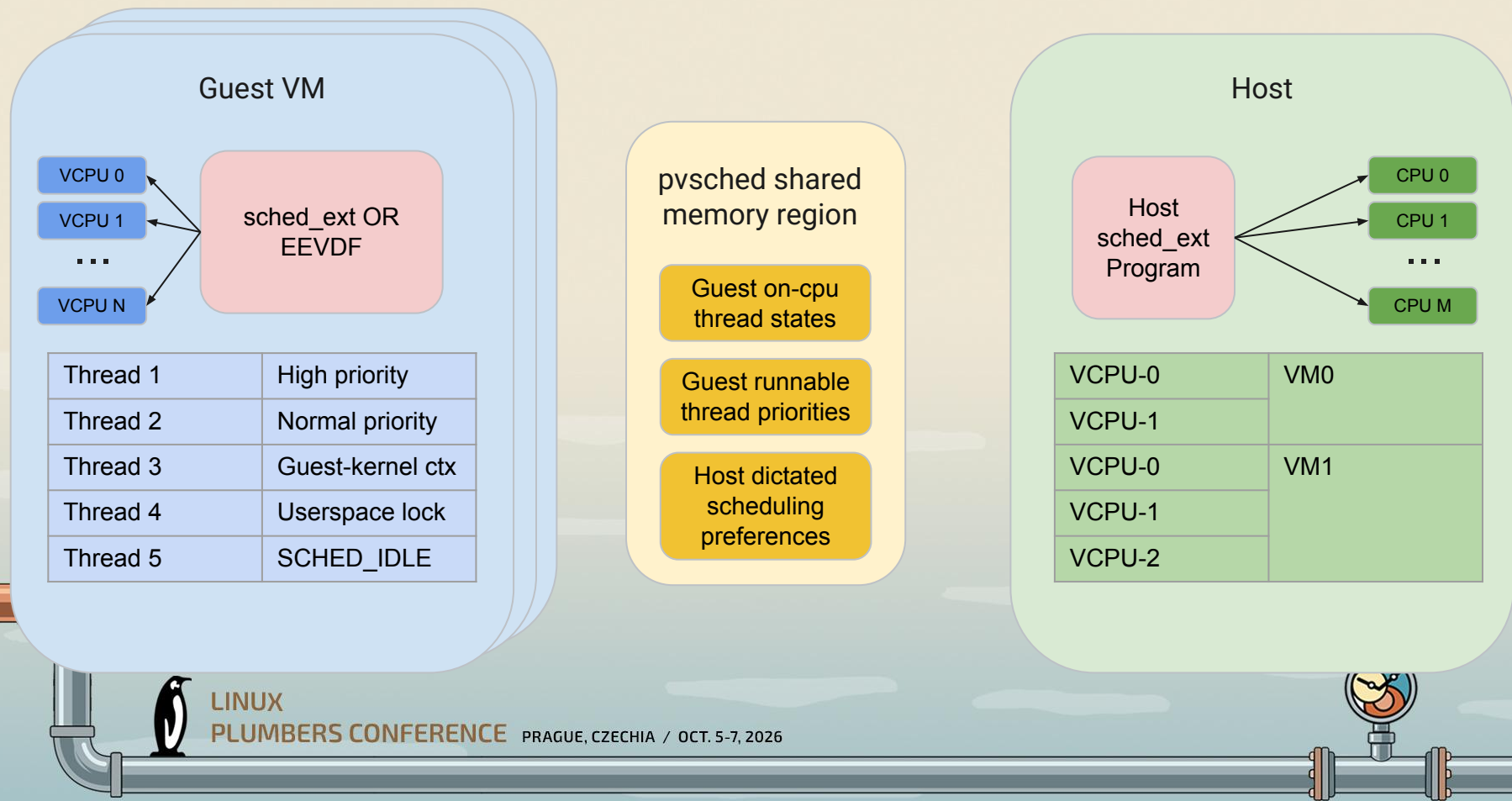
LINUX

PLUMBERS CONFERENCE

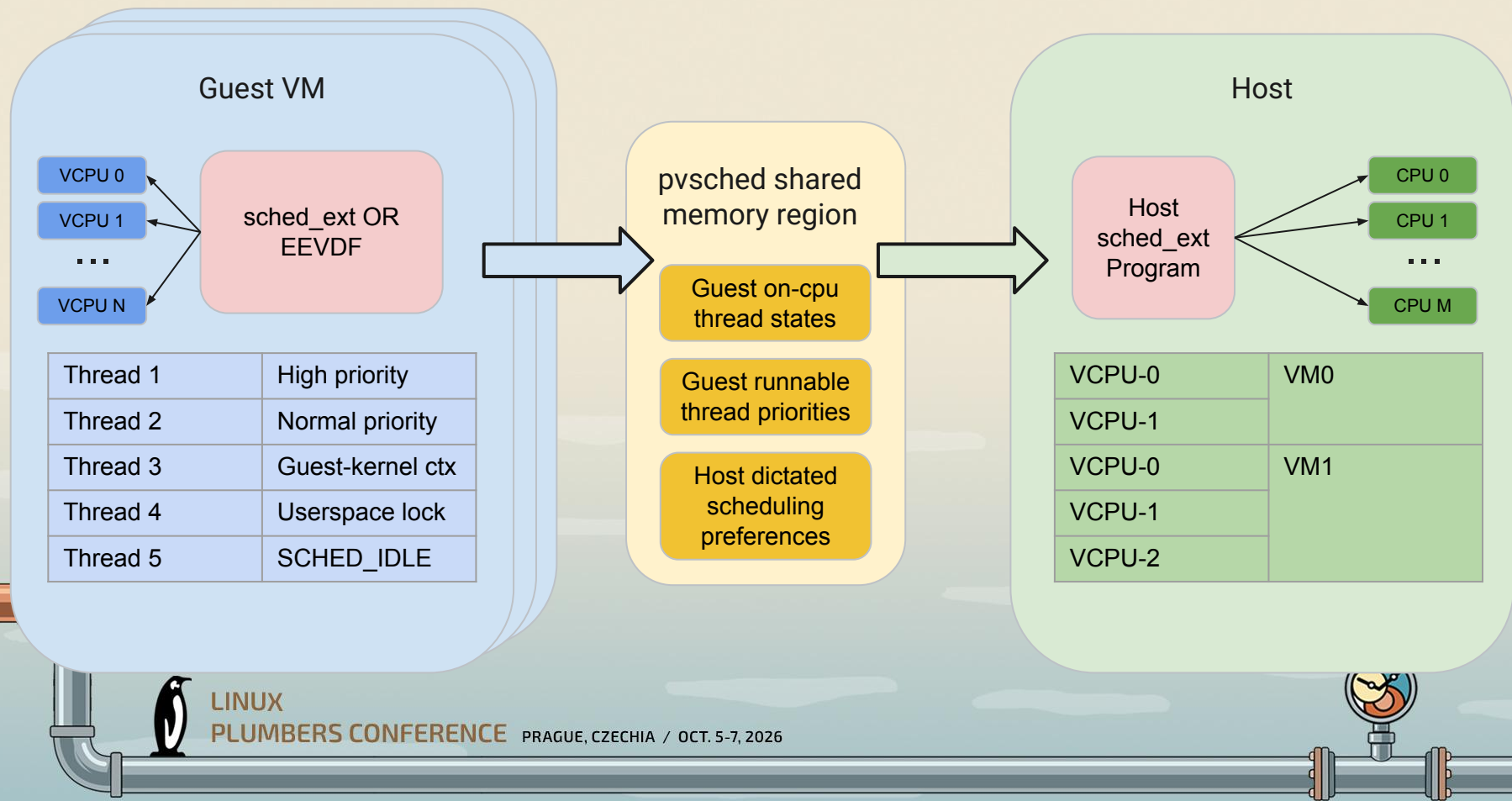
PRAGUE, CZECHIA / OCT. 5-7, 2026



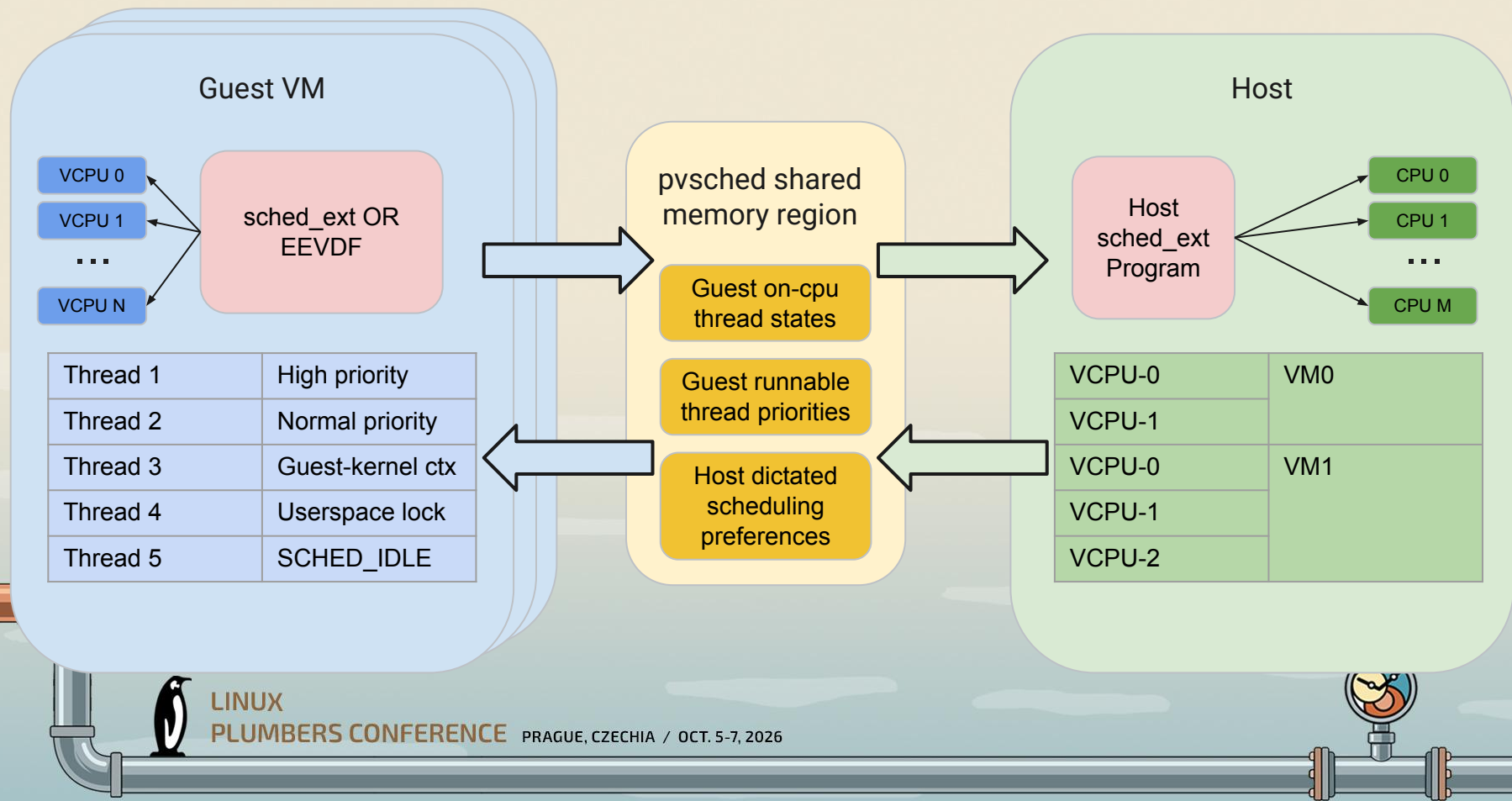
Paravirt scheduling communication



Paravirt scheduling communication



Paravirt scheduling communication



Integration with sched_ext

Host sched_ext Policy

Don't preempt sensitive VCPUs

- Guest IRQ
- High priority guest thread

Globally optimal decision making

- Gang schedule VCPUs
- Pre-stage intended target guest thread

Better fairness

- Temporarily break cgroup abstraction, for boost/deboost
- Adjust VM throughput based on dynamic internal load weight.



Integration with sched_ext

Host sched_ext Policy

Don't preempt sensitive VCPUs

- Guest IRQ
- High priority guest thread

Globally optimal decision making

- Gang schedule VCPUs
- Pre-stage intended target guest thread

Better fairness

- Temporarily break cgroup abstraction, for boost/deboost
- Adjust VM throughput based on dynamic internal load weight.

Guest sched_ext Policy

CPU funnelling

- Load balance towards VCPUs that are physically on CPU

Minimize VCPU spread under high host load conditions

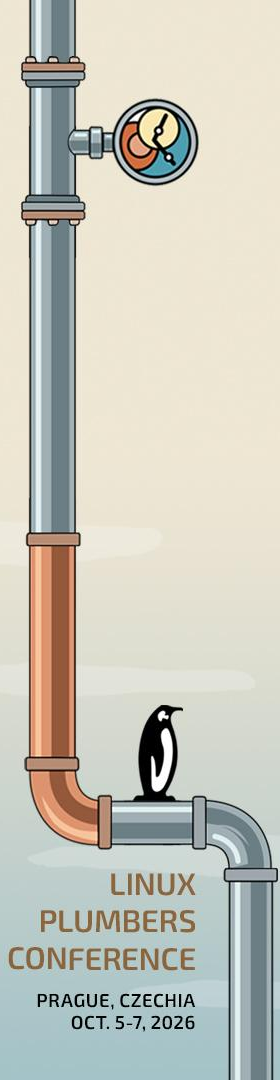
- Improve fairness and reduce VCPU overhead

Follow host hints

- Select which threads to schedule based on host guidance

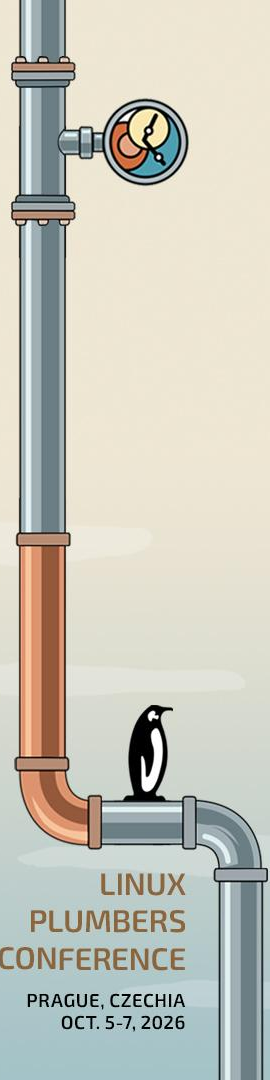


Beyond scheduling... pvshmem



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



Beyond scheduling... pvshmem

Reuse our kernel <> guest transport layer for broad policy communication.

FIRST TARGET: OOM KILL

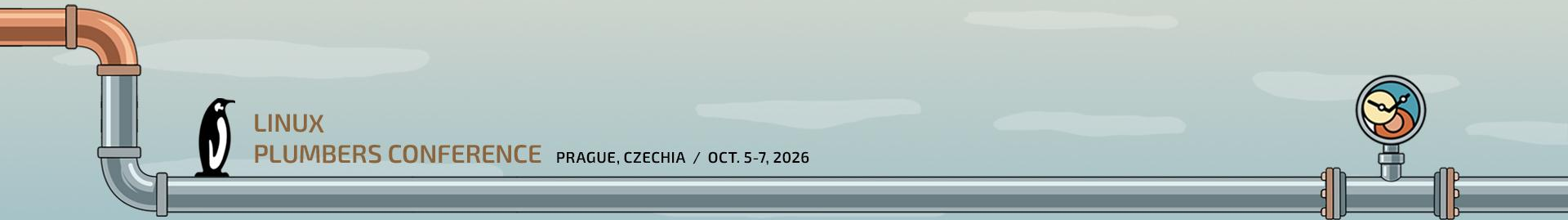
BPF-based OOM Management: Patches in progress to integrate eBPF policies.

Granular VM Control: OOM kill individual jobs within a VM rather than terminating the entire VM.

Guest Signals: Expose memcgs and memcg info (priority, usage, memory bounds) to host.

Host-Driven Actions: Host performs global victim selection and signals target guest to trigger OOM kill.

Thank you!



Backup: pvsched perf

Machine	p99 without	p99 with	≥ 1 ms: without → with
Intel N150, bare metal	1.15 ms	37–41 μ s	672–828 → 0–8
AMD Ryzen 6900HX, bare metal	1.3–1.8 ms	30–37 μ s	954–1083 → 2–4
Intel, nested VM	1.3–1.8 ms	41–76 μ s	≈ 850–1000 → 0–6

Test setup:

2-vcpu VM with vcpus pinned to physical cpus

VM cyclicttest:

```
cyclicttest -m -p 50 -t 2 -a 0-1 -i 1000
```

Host cpu hog:

```
stress-ng -cpu 1 to each pinned cpu
```



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

