

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Developing Next Generation Perf Tools With Python

Alice Rogers, BLSV

Ian Rogers, Google

Tracing Microconference

Outline

- Background
- Perf script interpreters
- Perf as a python module
- Examples:
 - ilist
 - flamegraphs
 - timechart
- What else can we build?
- Everything as a perf event



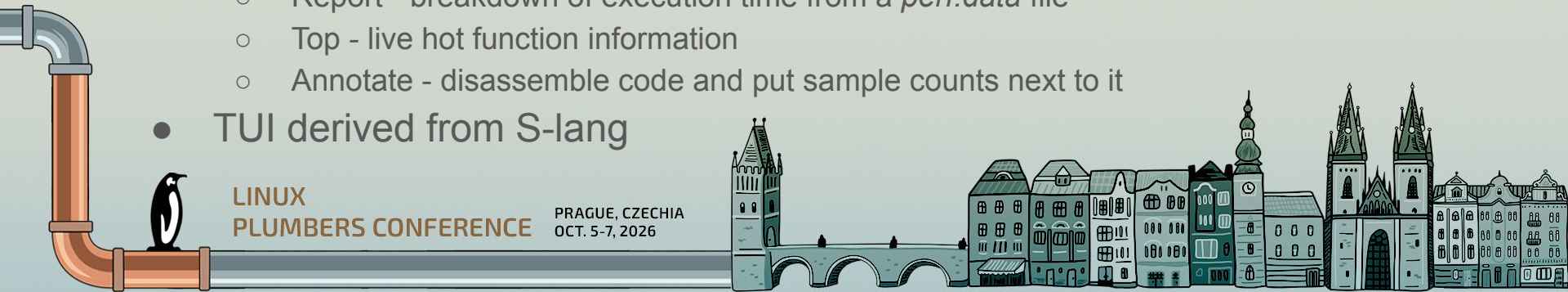
Linux Perf Brief Backstory

- Intel Pentium introduces performance counters. Libraries exposed the counters that could be read as root across the full system.
- 1999 - OProfile from John Levon at the University of Manchester
 - A driver that will save and restore performance counters on context switches
 - Ability to sample when a negatively programmed counter interrupts
- 2009 - Ingo Molnar and many others:
 - Performance Counters for Linux, v8 - <https://lkml.org/lkml/2009/6/6/149>
 - Deeper kernel integration - not a module
 - Extensibility and support for things like tracepoints
 - Performance
 - Active development



Inside the perf tool

- Code base derived from git
 - The perf sub-command implementations mirror the sub-command implementation in git
- Set of utilities for managing events, symbolization, etc.
- Main sub-commands:
 - List - list events
 - Stat - record event and metric counts
 - Record - event sampling, creates a *perf.data* file
 - Report - breakdown of execution time from a *perf.data* file
 - Top - live hot function information
 - Annotate - disassemble code and put sample counts next to it
- TUI derived from S-lang



Perf script and interpreters

- Perf script - displays a *perf.data* file
- Perl interpreter support added in [2009](#), Python in [2010](#)
- *trace_begin/trace_end* called at beginning and end of the file
- *process_event* and different handlers called for each event

```
def trace_begin():  
    parse_iomem()  
  
def trace_end():  
    print_memory_type()  
  
def process_event(param_dict):  
    if "sample" not in param_dict:  
        return  
  
    sample = param_dict["sample"]  
    if "phys_addr" not in sample:  
        return  
  
    phys_addr = sample["phys_addr"]  
    entry = find_memory_type(phys_addr)  
    if entry:  
        load_mem_type_cnt[entry] += 1  
  
    global event_name  
    if event_name is None:  
        event_name = param_dict["ev name"]
```

mem-phys-addr.py



Weaknesses of embedded interpreters

- With the embedded interpreters, perf becomes the interpreter
- The perf script loop is inherently single threaded
- Packages make assumptions that the main thread is the python interpreter
- For each *process_event* a dictionary is created with all possible values, whether needed or not



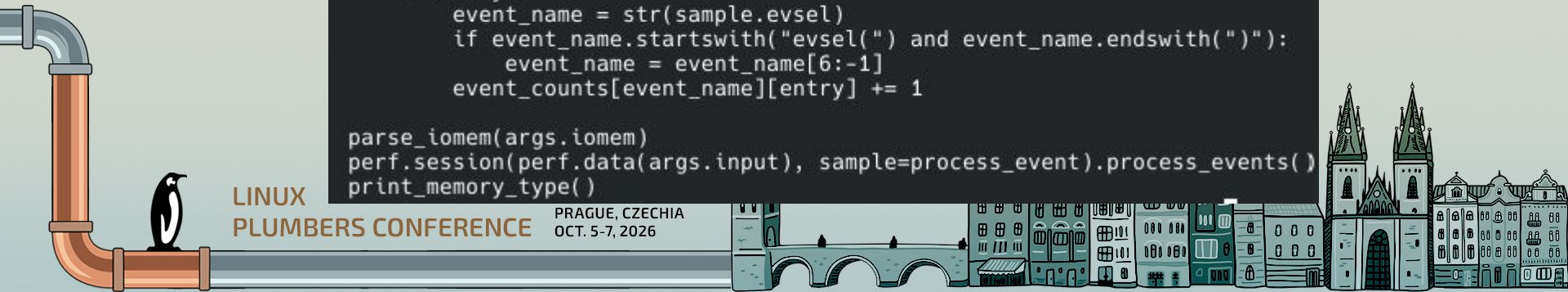
Outside rather than inside

- 2011 python bindings for perf - [lore](#)
 - C files built twice, once for python module and once for perf
 - Minimal amount of perf internals with stubs to break bindings
- 2024 turn perf into a series of libraries - [lore](#)
 - Python module built by linking separate perf libraries into the module excluding libraries like test and bench
- 2025 ilist - [lore](#)
 - expansion of module to support event parsing and browsing
- 2026 migrate perf scripts to python module - [lore](#)

```
def process_event(sample):  
    """Process a single sample event."""  
    phys_addr = sample.sample_phys_addr or 0  
    if not phys_addr:  
        return  
    entry = find_memory_type(phys_addr)  
    if entry:  
        event_name = str(sample.evsel)  
        if event_name.startswith("evsel(") and event_name.endswith(")"):   
            event_name = event_name[6:-1]  
        event_counts[event_name][entry] += 1  
  
parse_iomem(args.iomem)  
perf.session(perf.data(args.input), sample=process_event).process_events()  
print_memory_type()
```

LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



Outside rather than inside

- 2011 python bindings for perf - [lore](#)
 - C files built twice, once for python module and once for test module
 - Minimal amount of perf internals with stubs to support event
- 2024 turn perf into a series of libraries - [lore](#)
 - Python module built by linking separate perf libraries, excluding libraries like test and bench

Access state through attributes that are computed when used

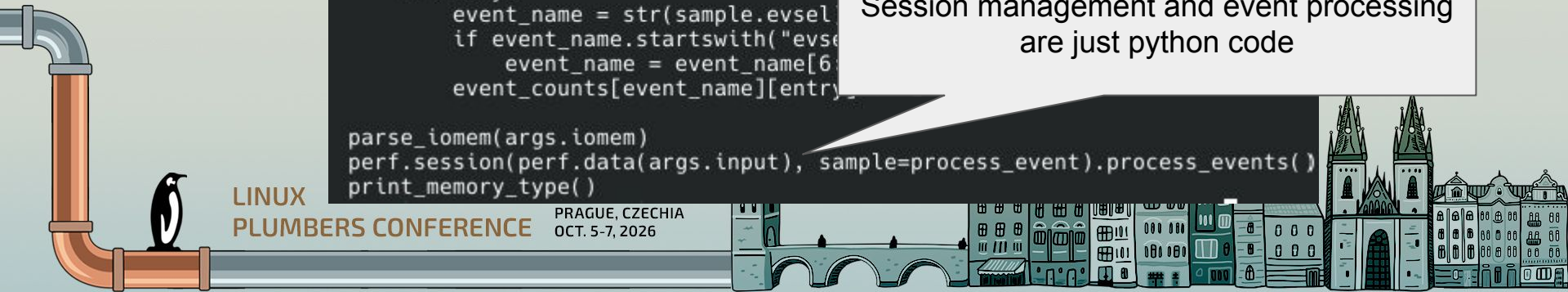
```
def process_event(sample):  
    """Process a single sample event."""  
    phys_addr = sample.sample_phys_addr or 0  
    if not phys_addr:  
        return  
    entry = find_memory_type(phys_addr)  
    if entry:  
        event_name = str(sample.evsel)  
        if event_name.startswith("evsel"):  
            event_name = event_name[6:]  
        event_counts[event_name][entry]
```

Session management and event processing are just python code

```
parse_iomem(args.iomem)  
perf.session(perf.data(args.input), sample=process_event).process_events()  
print_memory_type()
```

LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



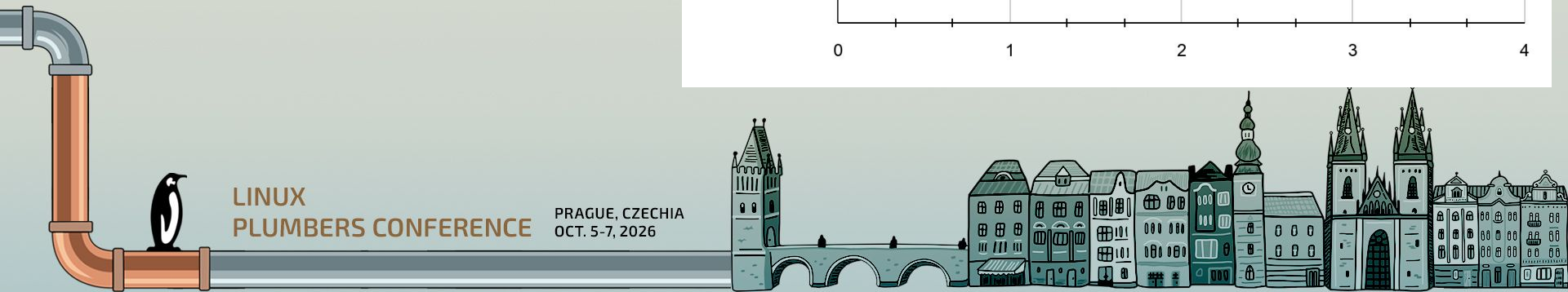
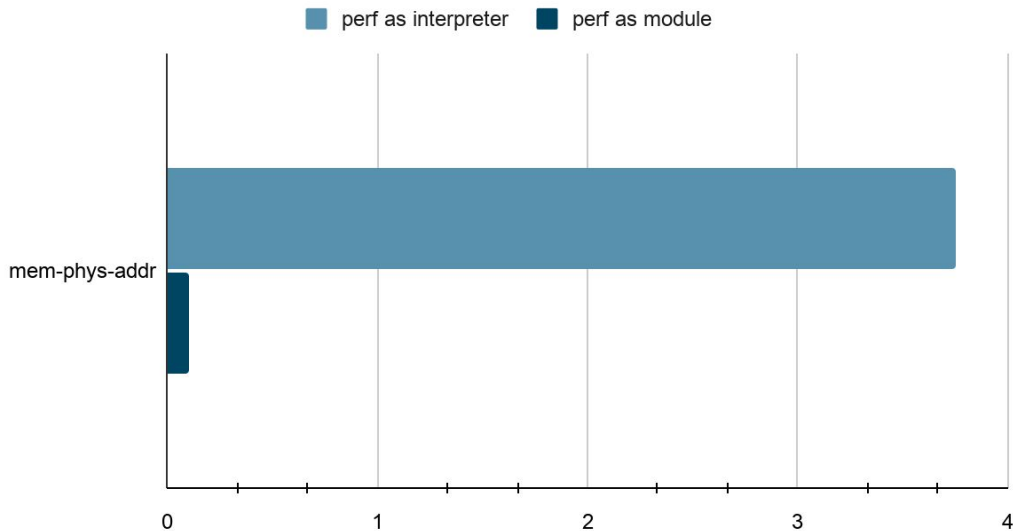
Performance

```
$ perf mem record -a sleep 1
```

```
$ perf script mem-phys-addr.py
```

35x improvement!

mem-phys-addr



ilist demo

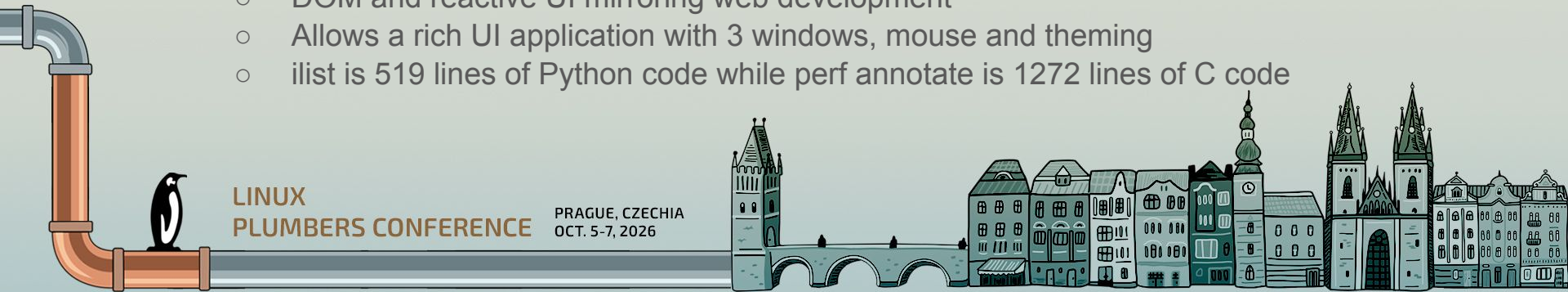


LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

TUI development shouldn't be hard

- S-lang in perf means
 - Low level event loops and screen layout
 - Lack of:
 - Mouse support
 - Theming
 - Widgets
 - Verbose and error prone code
- ilist in Python built on the Python textual framework
 - DOM and reactive UI mirroring web development
 - Allows a rich UI application with 3 windows, mouse and theming
 - ilist is 519 lines of Python code while perf annotate is 1272 lines of C code

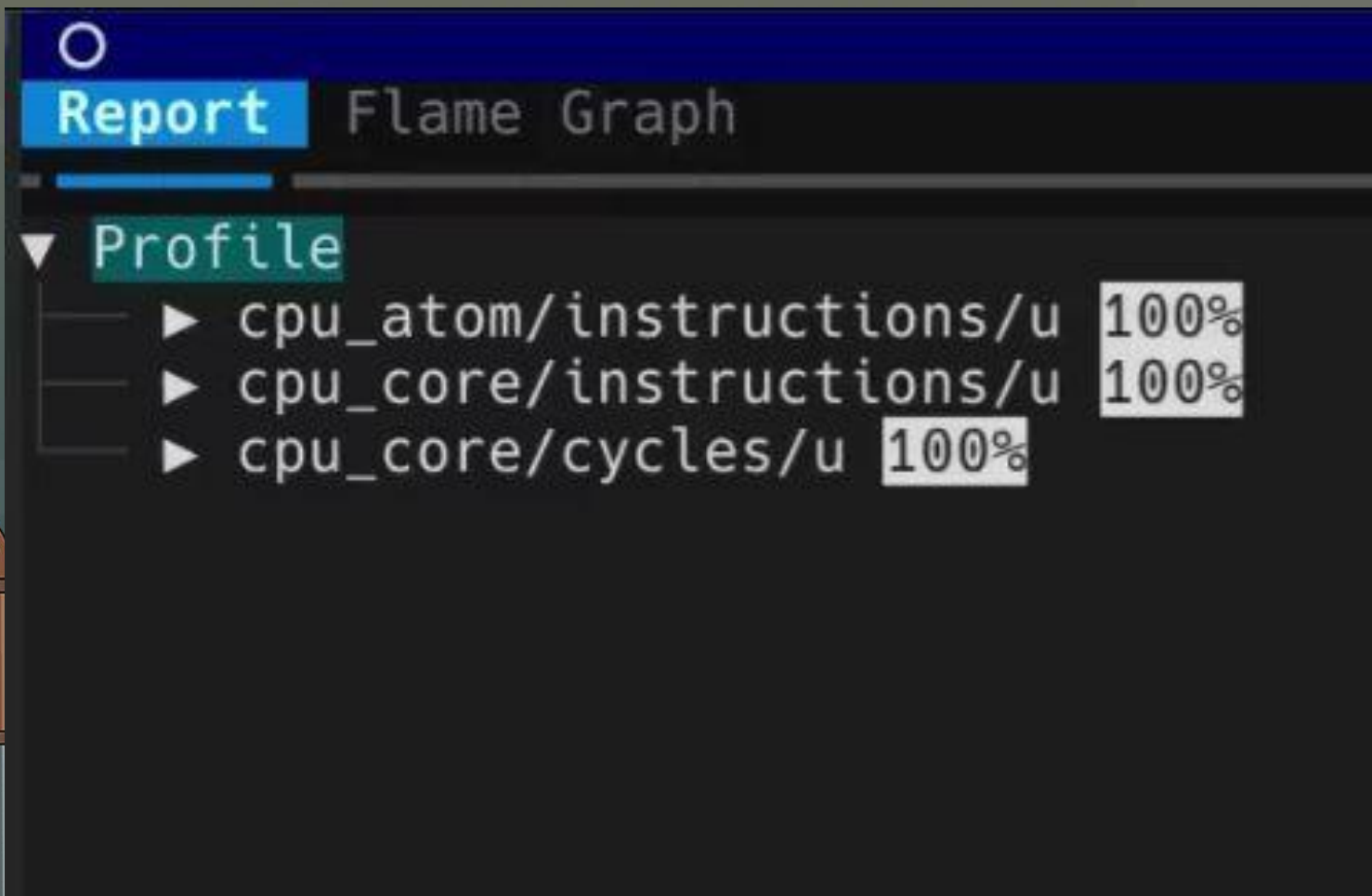


What can we build next?

- For visualizing profile data perf report is at the heart of the perf experience
- What would be next generation features of perf report?
 - Mouse support
 - Theming
 - Flame graphs?
 - Timecharts?
 - Interactive visualization while the file loads



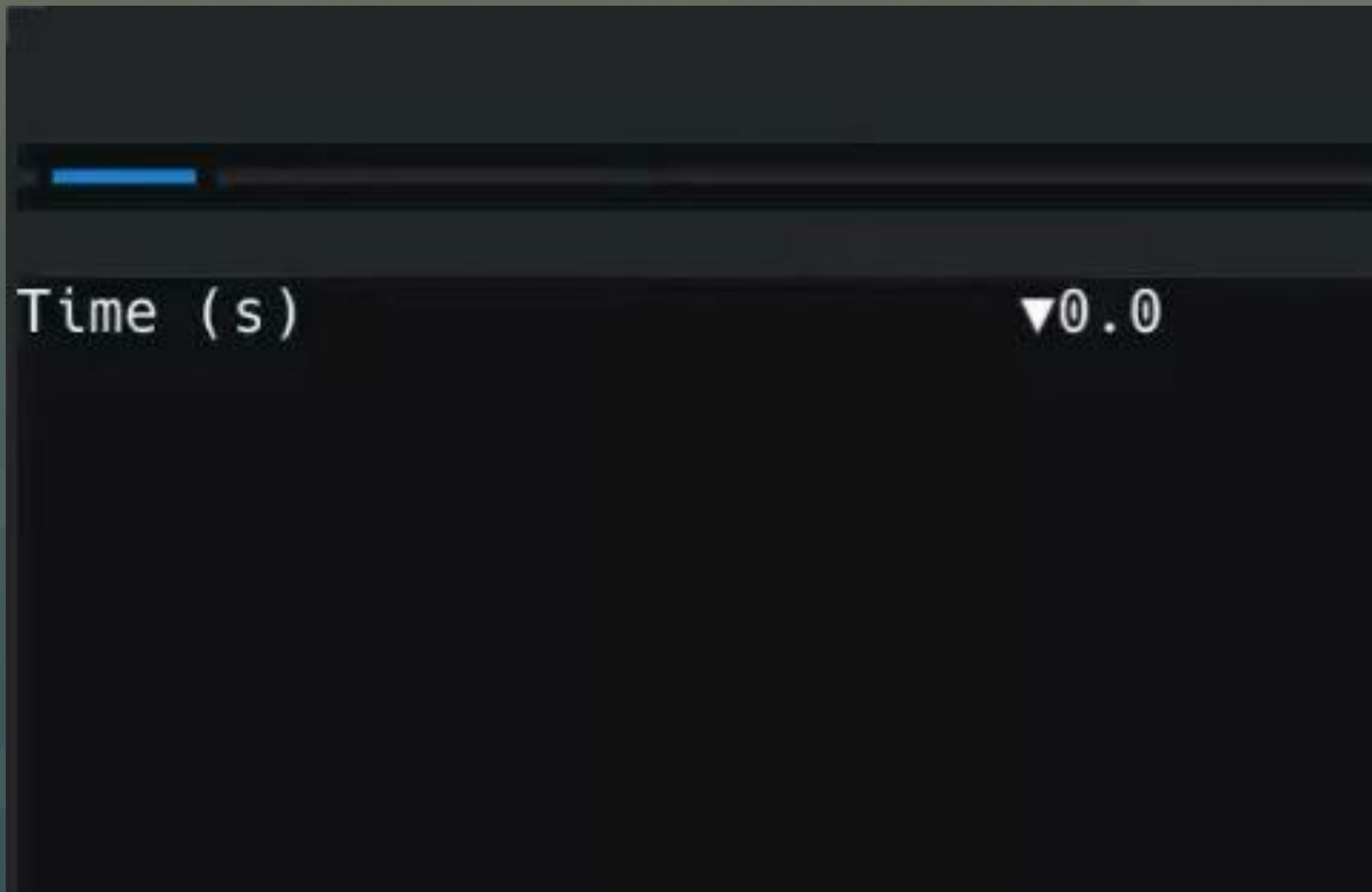
TUI flamegraph demo



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

TUI timechart demo



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

What's next

- LLM integration
 - Perf python module allows PMUs and events to be dynamically determined.
 - Live gathering of counters and sampling to determine performance bottlenecks
 - Topdown but beyond just CPU bottlenecks



Everything as a perf event

- Perf has blind spots in areas like:
 - Accelerators
 - Networking
 - Disk usage
- Build new PMUs into the kernel or in software
 - A perf tool software PMU can bridge existing libraries, subsystems into the perf API
 - Hwmon (landed - [lore](#)), DRM (landed - [lore](#)), Networking - [lore](#), NVMe - [lore](#), Memory - [lore](#)
- That's not to say the perf API is perfect but by working with the tool and drivers in the kernel tree we can evolve them together.



Conclusions and Thanks!

- Perf has transitioned from being a Perl/Python interpreter to being a fully featured Python module.
- The Python module has allowed new TUI development that comes on top of the legacy S-lang visualizations.
- Python is better suited to UI and LLM work than C.
- The more Perf events the richer the tooling becomes.

Thank you!

Question Time!



**LINUX
PLUMBERS
CONFERENCE**

PRAGUE, CZECHIA / OCT. 5-7, 2026