rbperf: Understanding Ruby with BPF

eBPF Summit

Javier Honduvilla Coto <javierhonduco@fb.com>

October 28th-29th, 2020

Why BPF?

Why BPF?

- Flexibility

Why BPF?

- Flexibility
- Low overhead

Why BPF?

- Flexibility
- Low overhead
- Continuous profiling

Why BPF?

- Flexibility
- Low overhead
- Continuous profiling
- No modifications of the tracee

rbperf

rbperf

- Profile Ruby programs

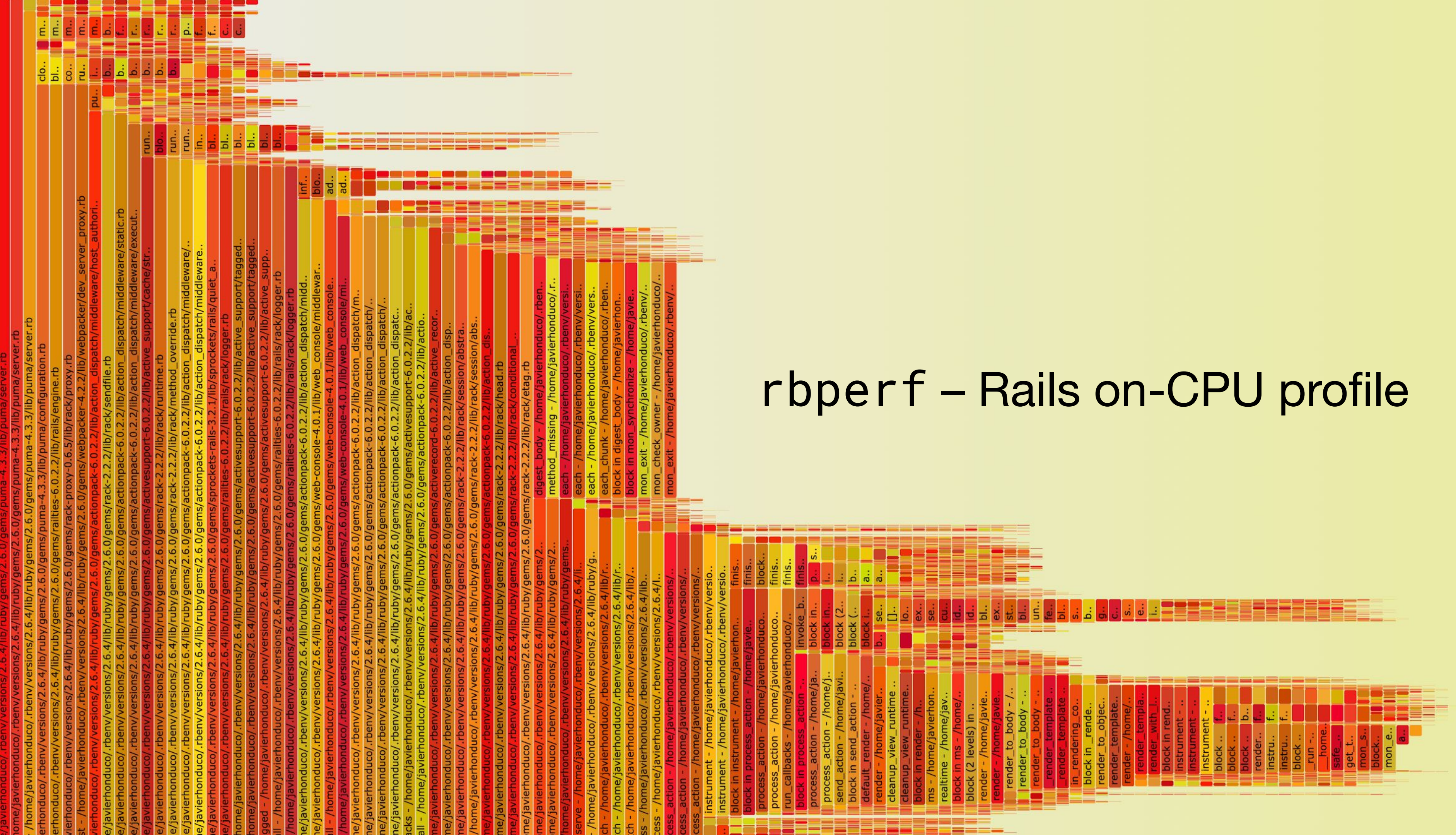
rbperf

- Profile Ruby programs
- Trace complex Ruby programs execution

rbperf – on-CPU profiling

- `$ rbperf record --pid=124 cpu`

- `$ rbperf report [...]`



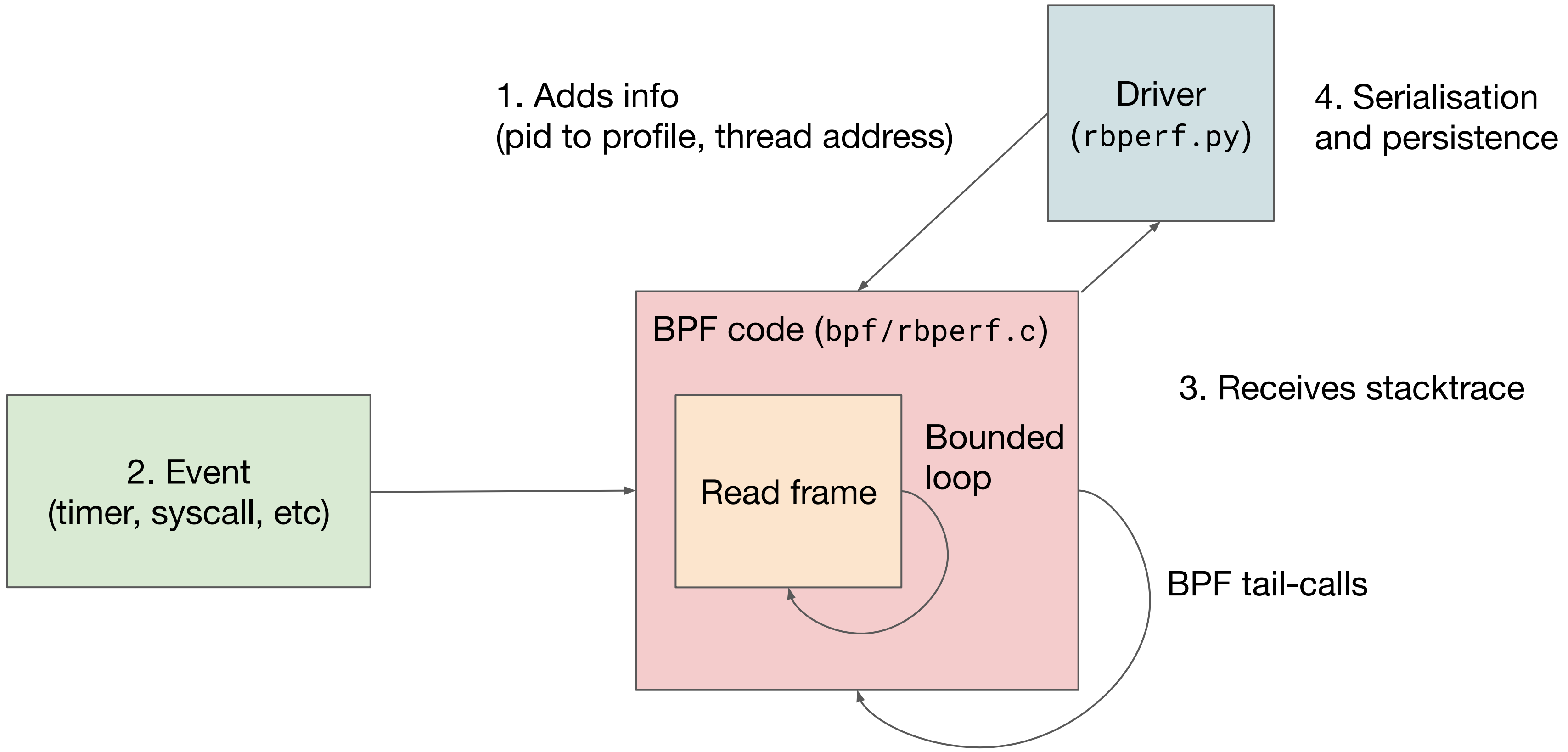
`rbperf` – Rails on-CPU profile

`rbperf` – tracing `write(2)` calls

```
-  $ rbperf record \
     --pid=124 event \
     --tracepoint=syscalls:sys_enter_write
```

```
-  $ rbperf report [...]
```

Architecture



Challenges

- Implementing the stack walking for a dynamic language

Challenges

- Implementing the stack walking for a dynamic language
- Supporting multiple Ruby versions

Challenges

- Implementing the stack walking for a dynamic language
- Supporting multiple Ruby versions
- Correctness testing

Challenges

- Implementing the stack walking for a dynamic language
- Supporting multiple Ruby versions
- Correctness testing
- BPF safety features

Future plans

- Integrate in Facebook’s profiling infra
- Rewrite OSS driver program
- Make the OSS version awesome
    - Better documentation (including how to measure overhead)
    - Add more output formats
    - Open source GDB / drgn helper
    - Other tools?
    - Containers support?
    - Support request-oriented workloads?

Thanks! :)

https://github.com/**facebookexperimental/rbperf**

javierhonduco@fb.com

@javierhonduco