3rd Workshop on eBPF and Kernel Extensions (eBPF)

No Two Snowflakes Are Alike: Studying eBPF Libraries' Performance, Fidelity and Resource Usage

Carlos Machado[1], Bruno Gião[1], Sebastião Amaro[2], Miguel Matos[2], João Paulo[1] and Tânia Esteves[1]

[1] INESC TEC & University of Minho, [2] IST Lisbon & INESC-ID

eBPF Library Landscape

- Facilitate writing, loading and managing eBPF programs
- Since 2015 (BCC), multiple libraries with different features have emerged

eBPF Library Landscape

- Facilitate writing, loading and managing eBPF programs
- Since 2015 (BCC), multiple libraries with different features have emerged

How can we choose the **right one**?

Motivation - Choosing an eBPF Library

- Most current eBPF studies focus on the technology or use cases, with little attention to the libraries themselves [1,2]
- Existing comparisons focus only on qualitative metrics (e.g. programming language, portability, ease of use) [3,4]

[1] - Marcos Vieira et al. “Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications”, 2020.
[2] - H. Sharaf, I. Ahmad and T. Dimitriou, “Extended Berkeley Packet Filter: An Application Perspective”, 2022.
[3] - Rice, Liz. “Learning eBPF”. O'Reilly Media, Inc., 2023.
[4] - eBPF Chirp, Substack. “Go, C, Rust, and More: Picking the Right eBPF Application Stack”, 2025.

Motivation - Choosing an eBPF Library

- Most current eBPF studies focus on the technology or use cases, with little attention to the libraries themselves [1,2]
- Existing comparisons focus only on qualitative metrics (e.g. programming language, portability, ease of use) [3,4]

What about the **quantitative metrics** (e.g. performance and resource usage)?

[1] - Marcos Vieira et al. “Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications”, 2020.
[2] - H. Sharaf, I. Ahmad and T. Dimitriou, “Extended Berkeley Packet Filter: An Application Perspective”, 2022.
[3] - Rice, Liz. “Learning eBPF”. O'Reilly Media, Inc., 2023.
[4] - eBPF Chirp, Substack. “Go, C, Rust, and More: Picking the Right eBPF Application Stack”, 2025.

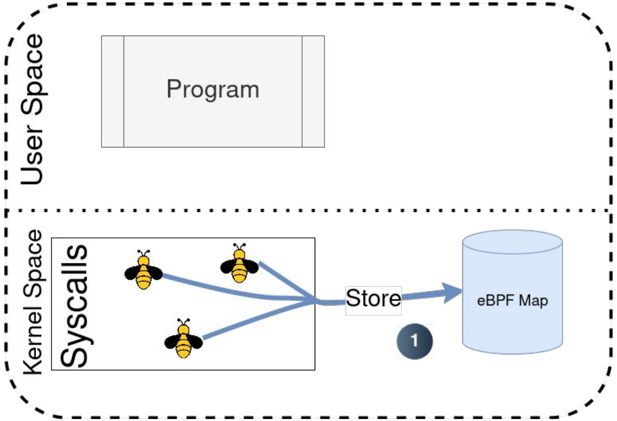
Research Goal

Efficiency analysis of popular eBPF libraries

- Performance
  - Impact on throughput, latency and runtime
- Resource Usage
  - Overhead on CPU, RAM and energy usage
- Fidelity
  - Capability to accurately capture events (i.e., without event loss)

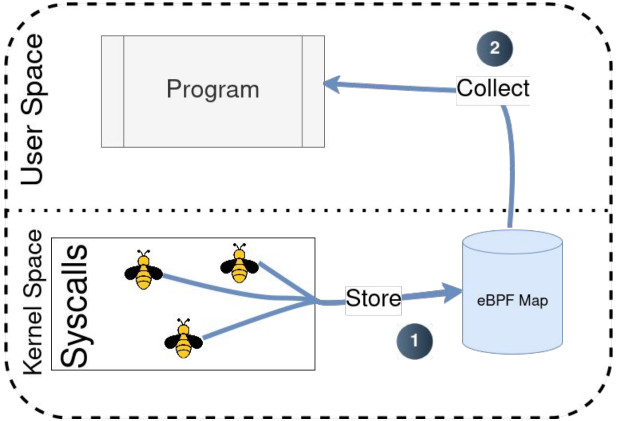
Methodology - Tools Implemented

syscount (lightweight)



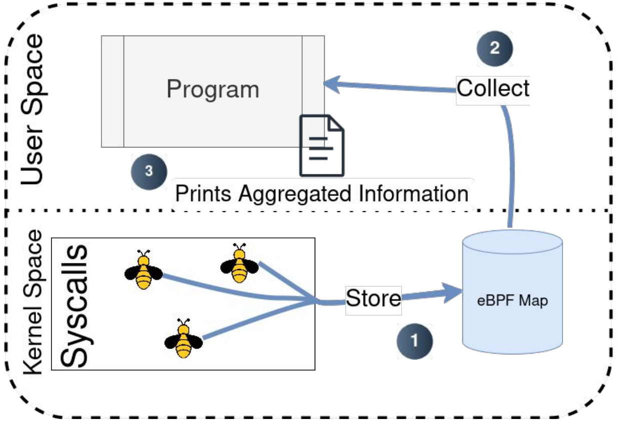
Methodology - Tools Implemented

syscount (lightweight)



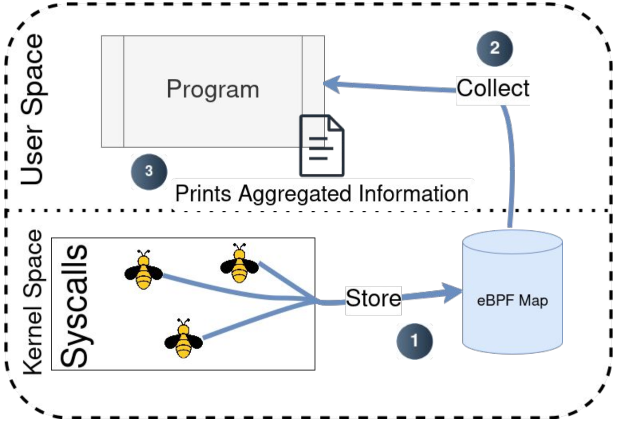
Methodology - Tools Implemented

syscount (lightweight)

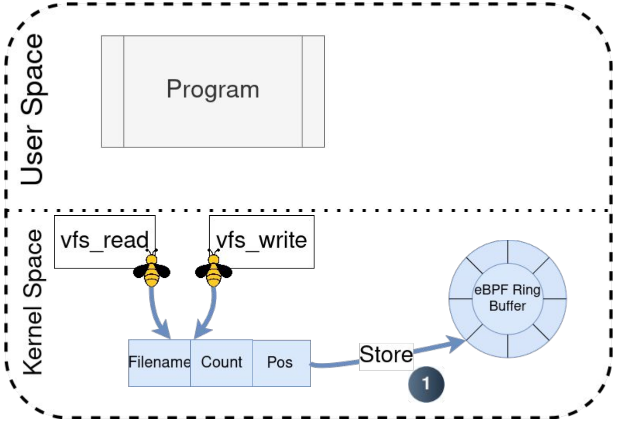


Methodology - Tools Implemented

syscount (lightweight)

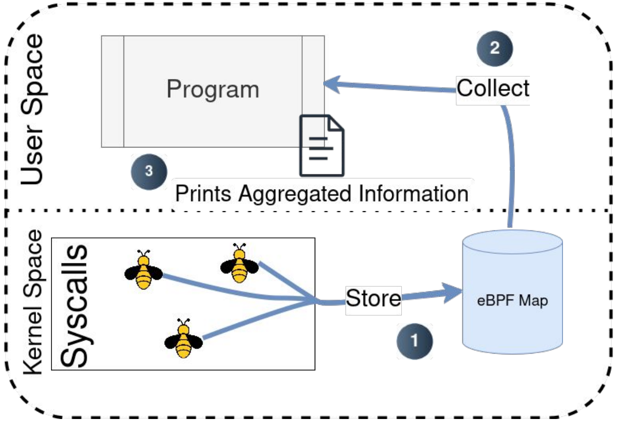


rw-tracer (moderate)

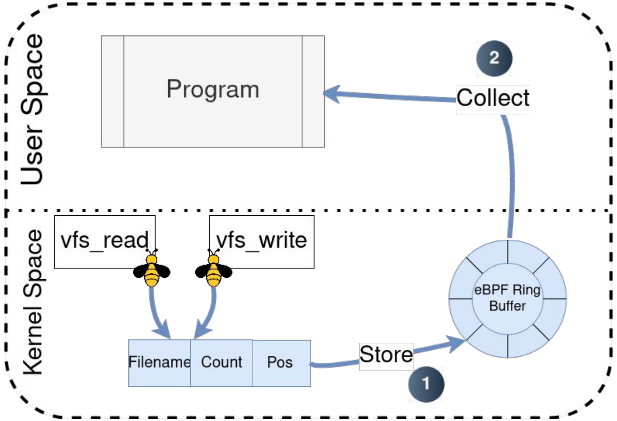


Methodology - Tools Implemented

syscount (lightweight)

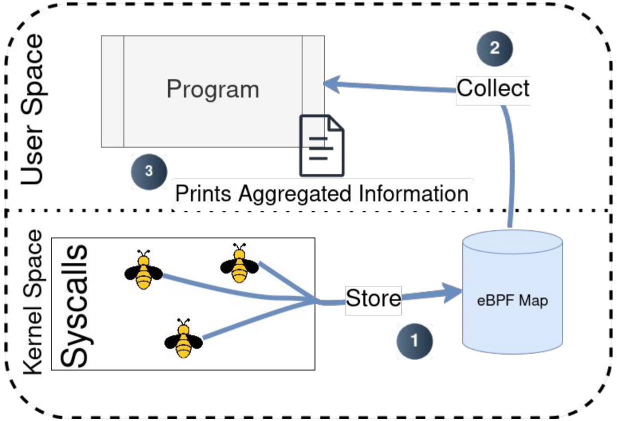


rw-tracer (moderate)

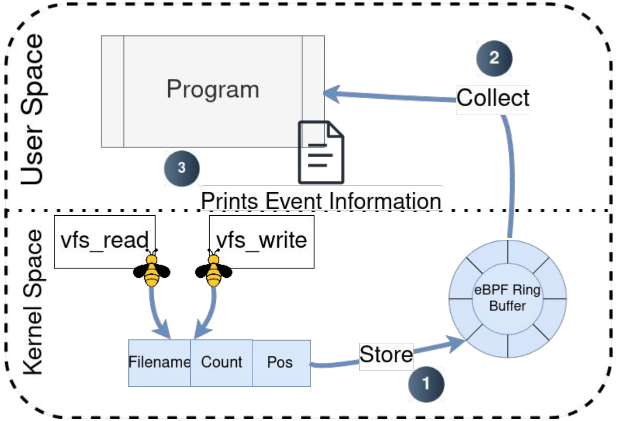


Methodology - Tools Implemented

syscount (lightweight)

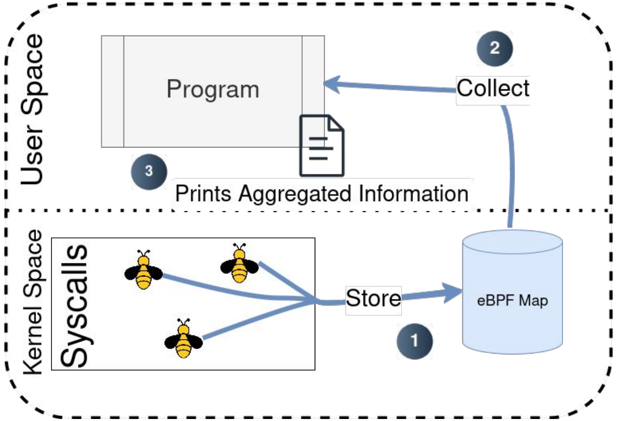


rw-tracer (moderate)

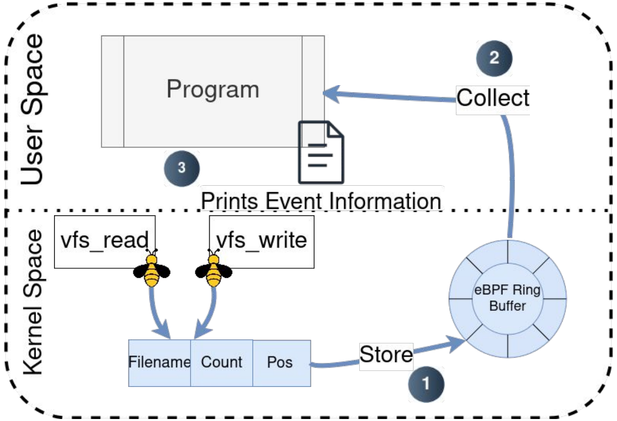


Methodology - Tools Implemented

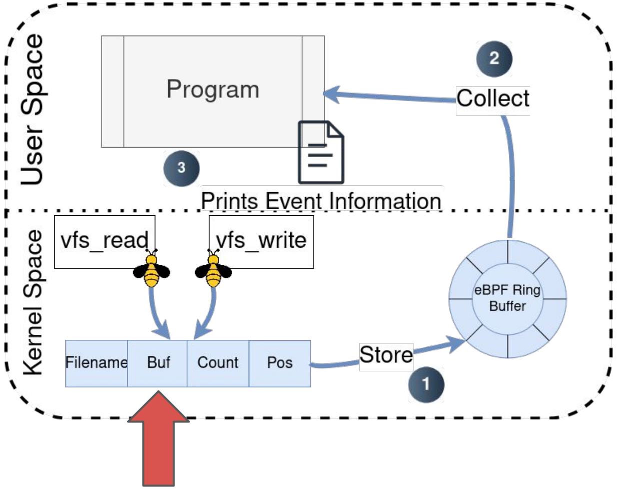
syscount (lightweight)



rw-tracer (moderate)

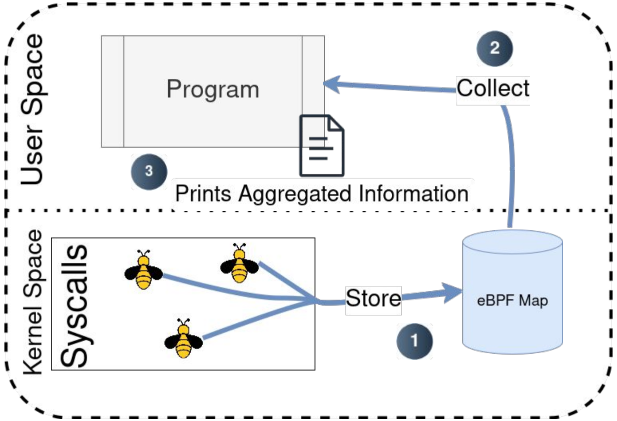


rw-tracer-all (more intensive)

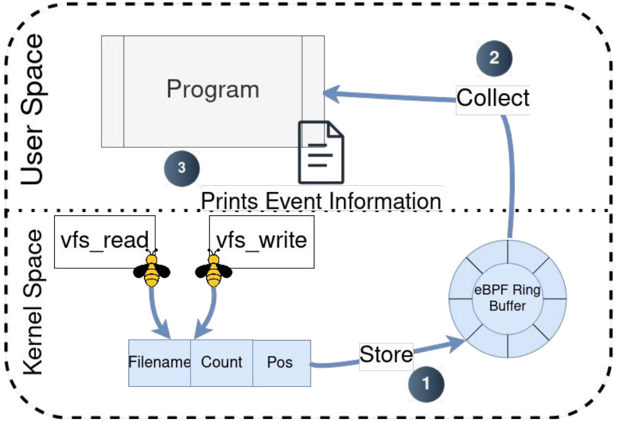


Methodology - Tools Implemented

syscount (lightweight)

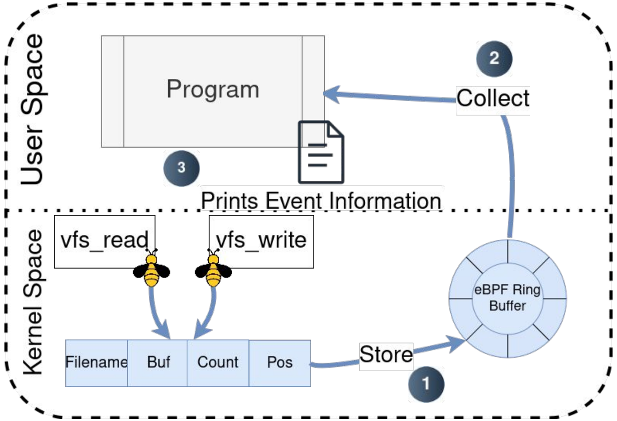


rw-tracer (moderate)



≈156 B per event

rw-tracer-all (more intensive)



≈4.2 KB per event

Methodology - eBPF Libraries Used

	BCC	bpftrace	libbpf	ebpf-go	Aya
User space	Python	Scripting Language	C	Go	Rust
Kernel	C		C	C	Rust

- **AyaSync -** Synchronous + custom (active) polling
- **AyaAsync -** Asynchronous + epoll (via AsyncFd)

Methodology - Experimental Setup and Metrics

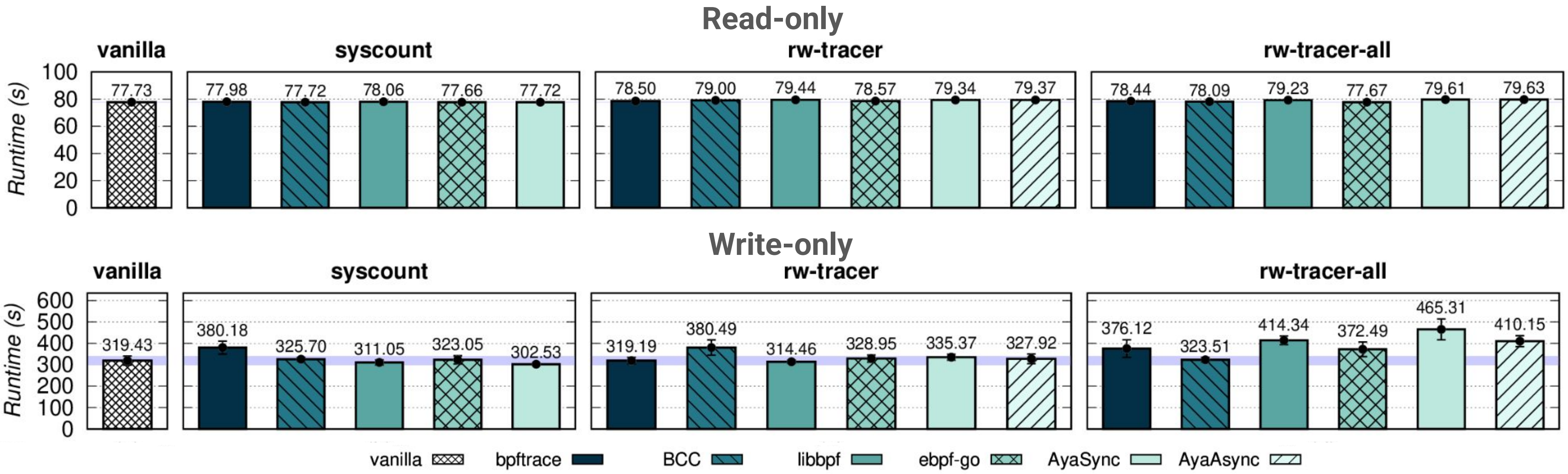
- **Five identical servers** (one per library)
  - Intel® Core™ i5-9500 @ 3.00 GHz, with 6 cores
  - 16 GiB RAM
  - 500 GiB SATA HDD + 240 GiB NVMe SSD
  - Ubuntu 24.04, kernel version 6.8.0-58-generic

- **Resource Monitoring**
  - CPU and memory (Dstat)
  - Energy (Intel RAPL)
- **Fidelity**
  - Total vs lost events
- **Workloads**
  - FIO benchmark for read, write, mixed 50/50 workloads, generating ≈33 M events
  - Runtime, throughput and latency

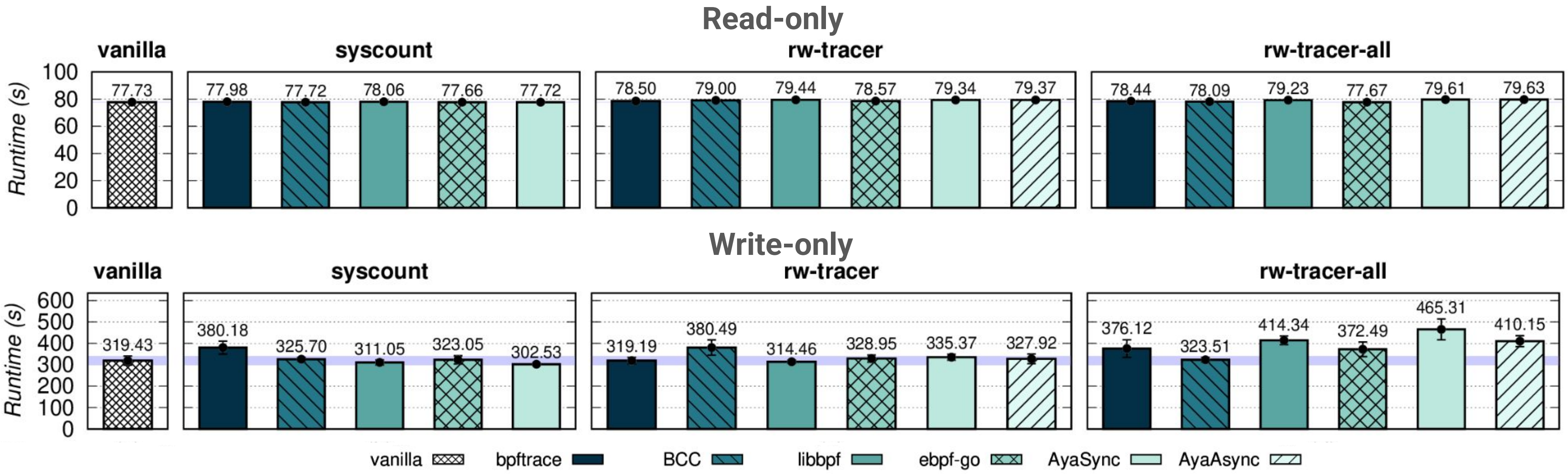
Each experiment repeated 3 times (average and standard deviation)

Results compared against a vanilla setup not using eBPF

Experimental Results - Performance Overhead

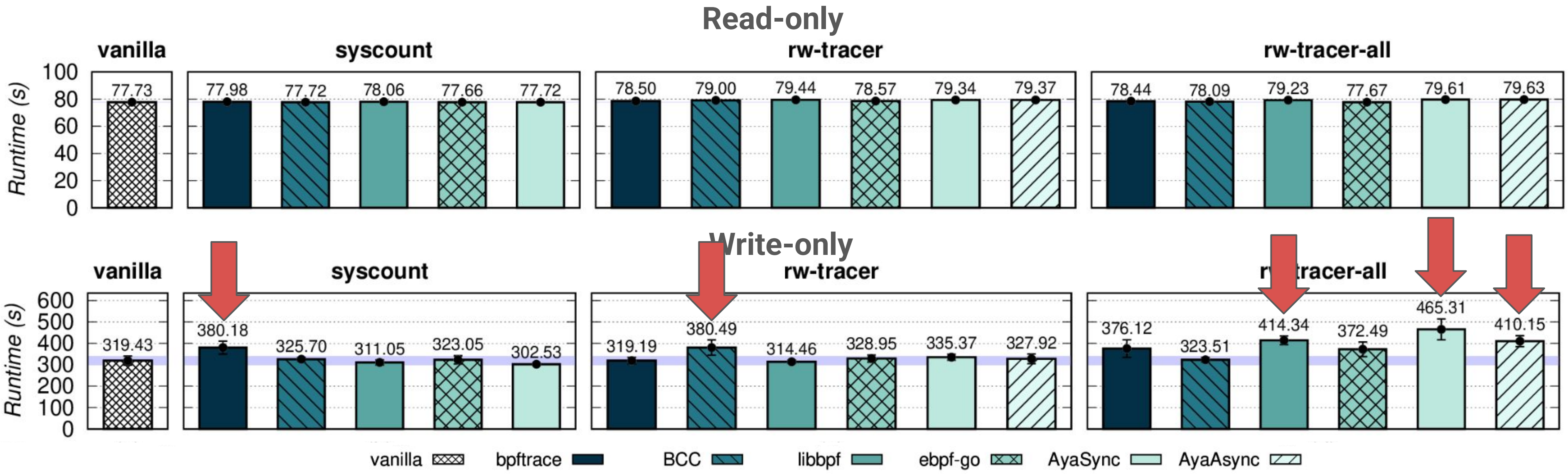


Experimental Results - Performance Overhead



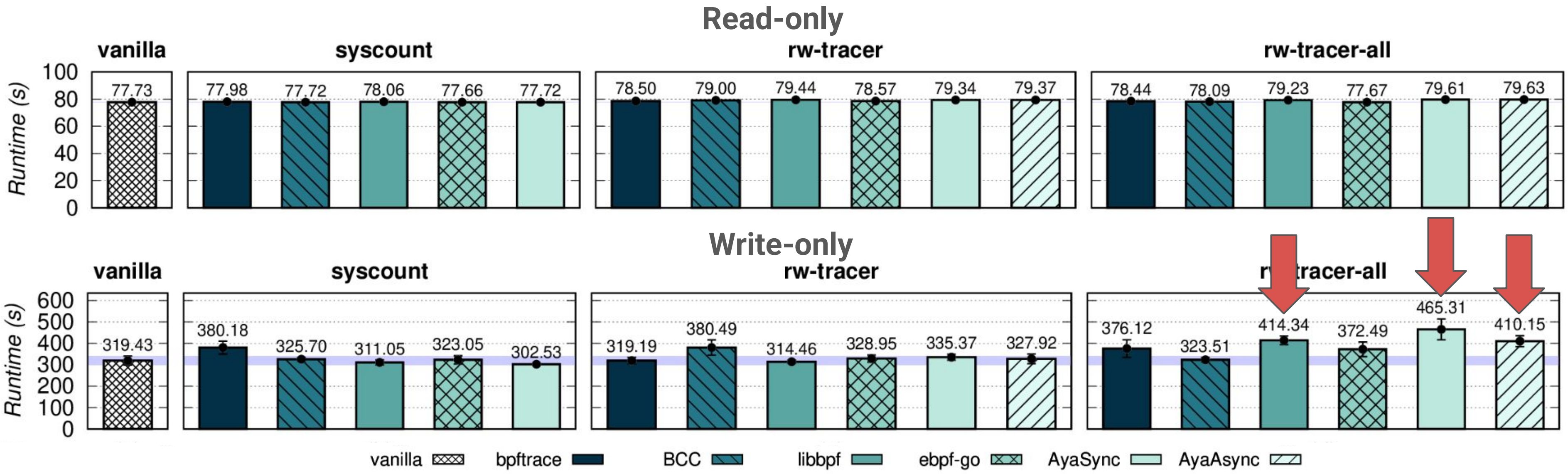
- Depends on workload event generation rate
  - Aya and libbpf introduce the highest overhead

Experimental Results - Performance Overhead



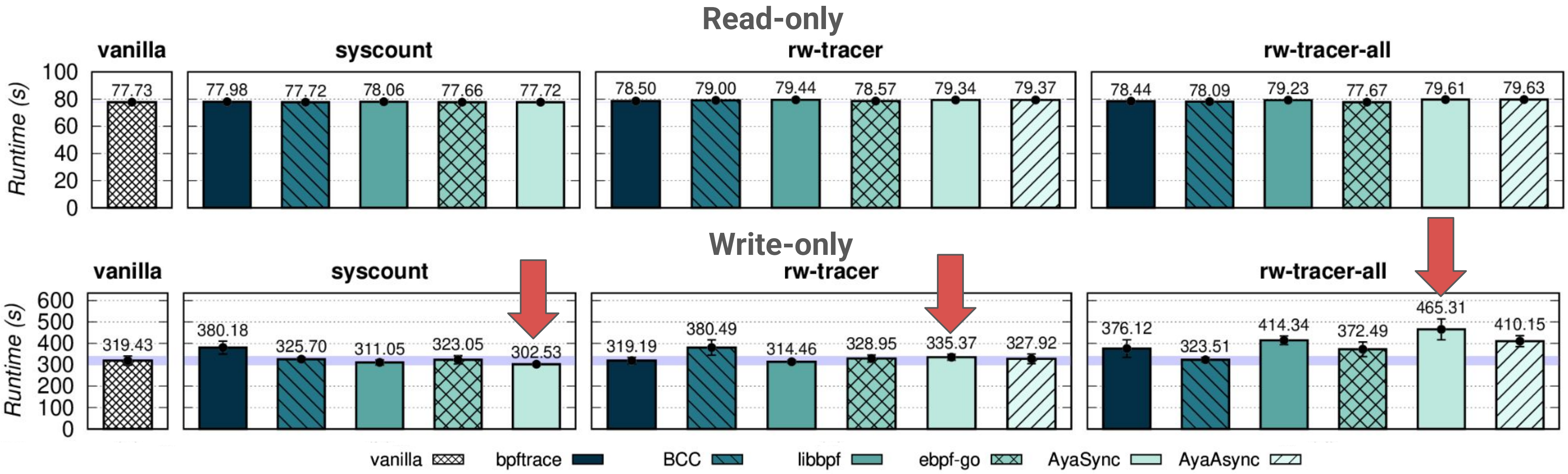
- Depends on workload event generation rate
    - Aya and libbpf introduce the highest overhead

Experimental Results - Performance Overhead



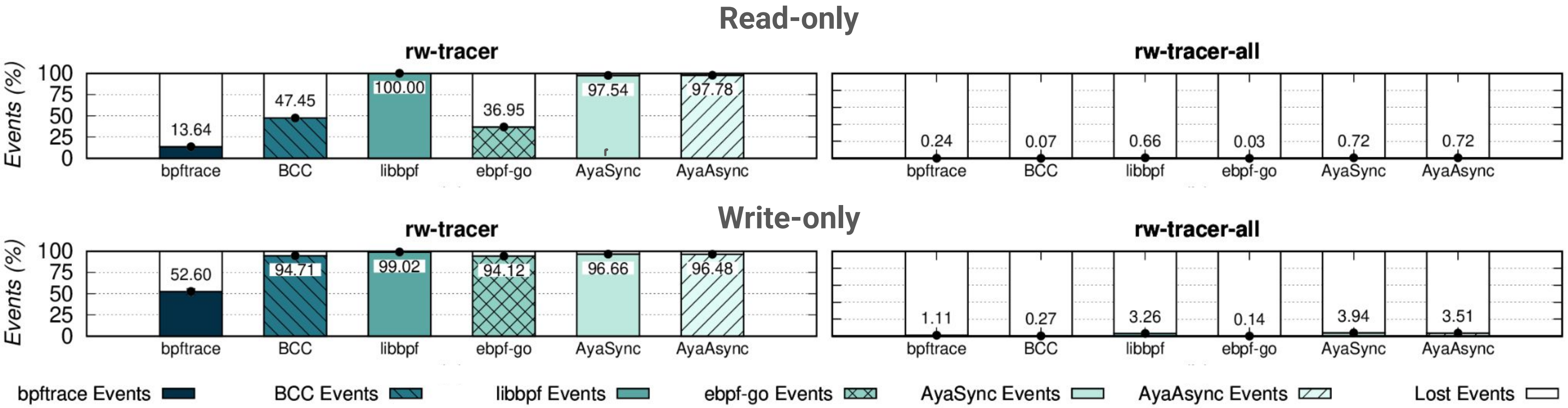
- Depends on workload event generation rate
  - Aya and libbpf introduce the highest overhead

Experimental Results - Performance Overhead

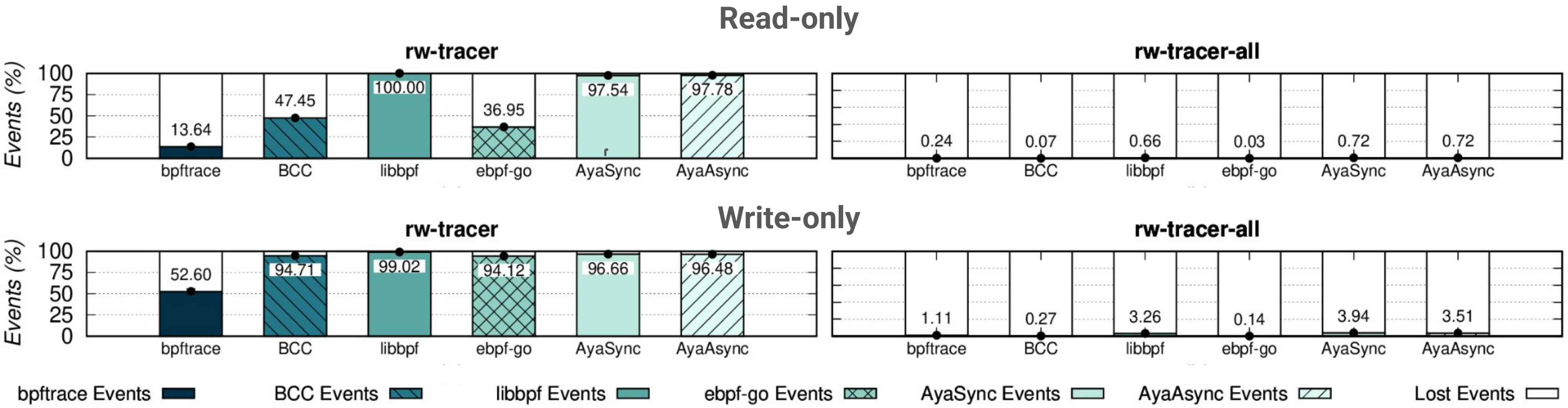


- Depends on workload event generation rate
  - Aya and libbpf introduce the highest overhead
- Impacted by the complexity of the eBPF tool
  - Heavier tools like rw-tracer-all introduced higher overheads

Experimental Results - Fidelity

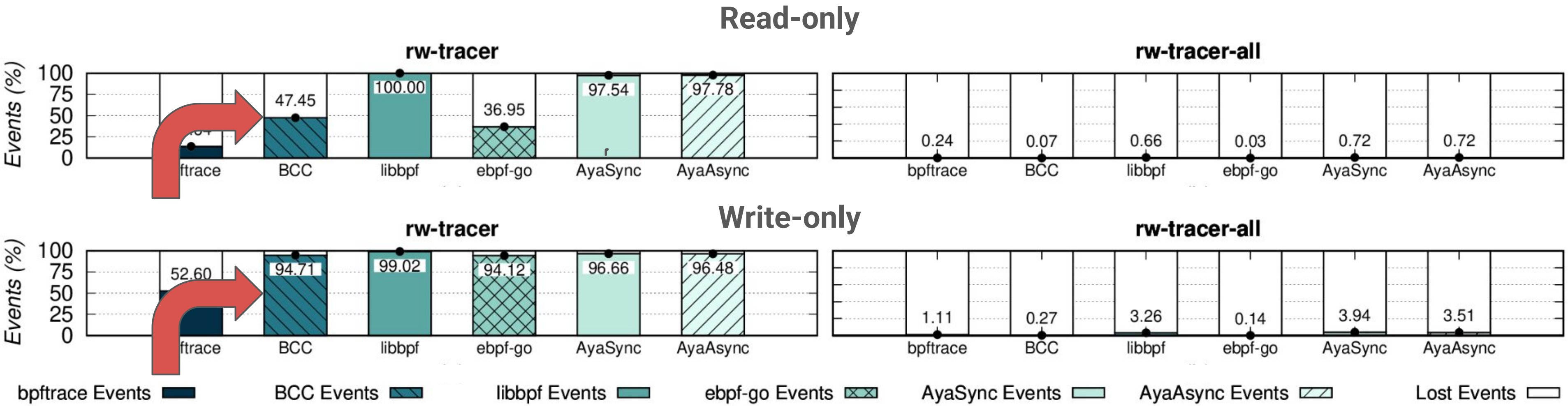


Experimental Results - Fidelity



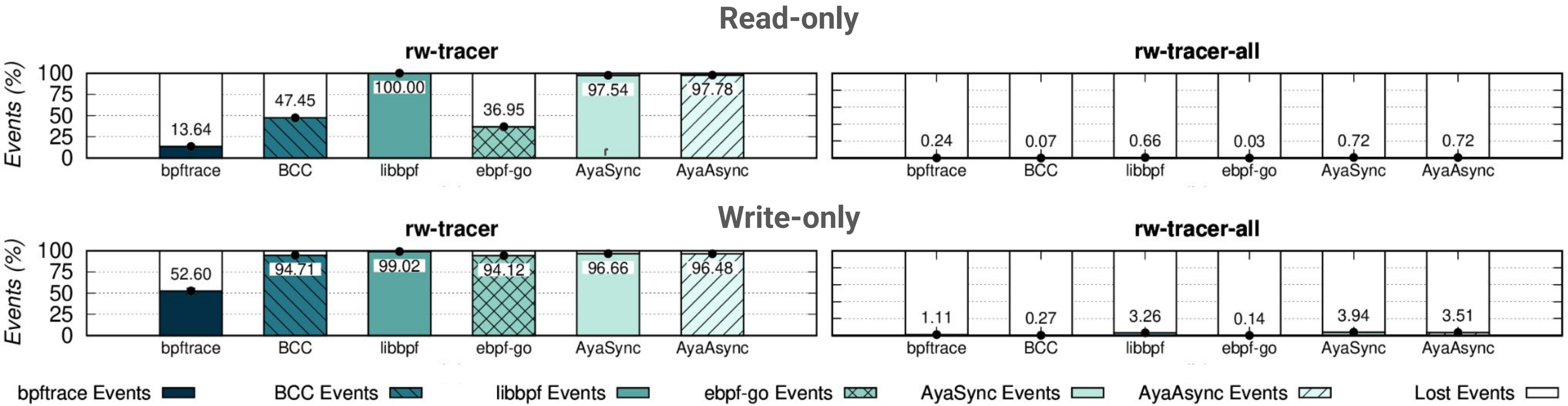
- Impacted by the workload event generation rate
  - Faster workloads like read-only cause more event loss overall

Experimental Results - Fidelity



- Impacted by the workload event generation rate
  - Faster workloads like read-only cause more event loss overall

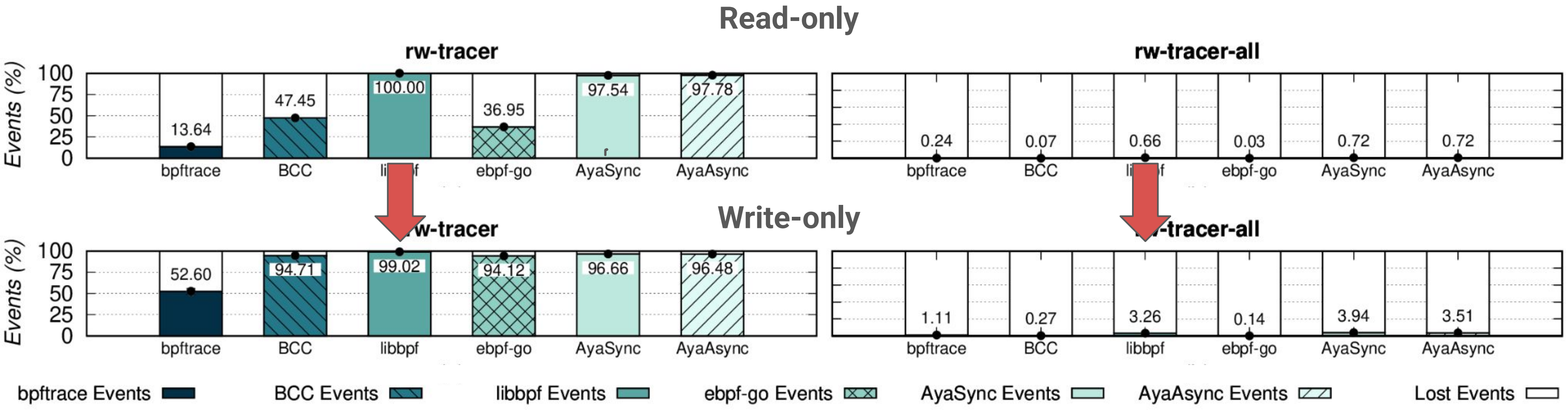
Experimental Results - Fidelity



- Impacted by the workload event generation rate
  - Faster workloads like read-only cause more event loss overall

- Also influenced by the amount of data sent to user space
  - While rw-tracer captures most of the events, rw-tracer-all loses most of them

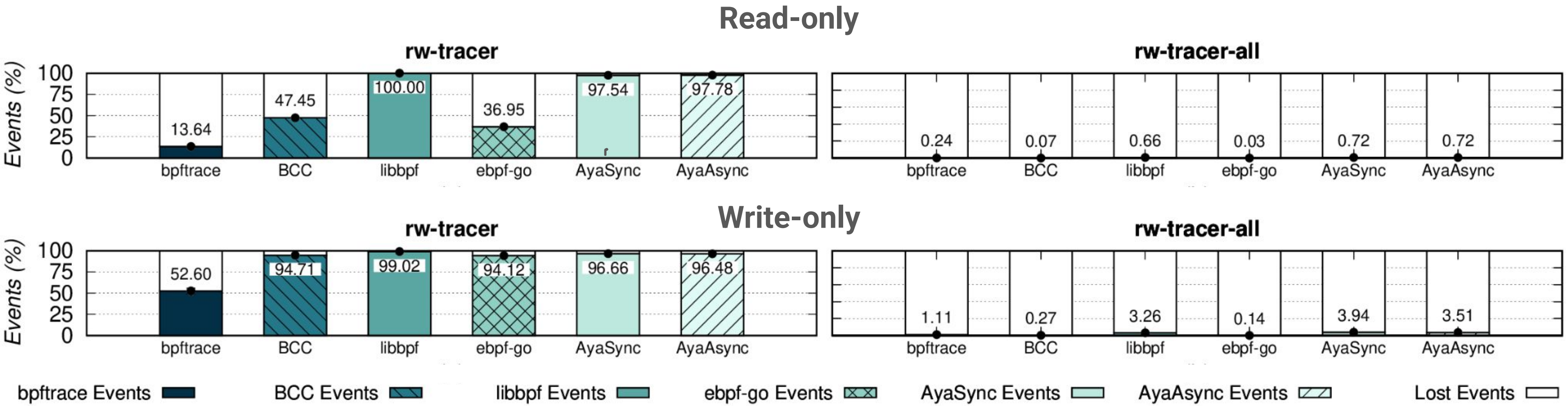
Experimental Results - Fidelity



- Impacted by the workload event generation rate
    - Faster workloads like read-only cause more event loss overall

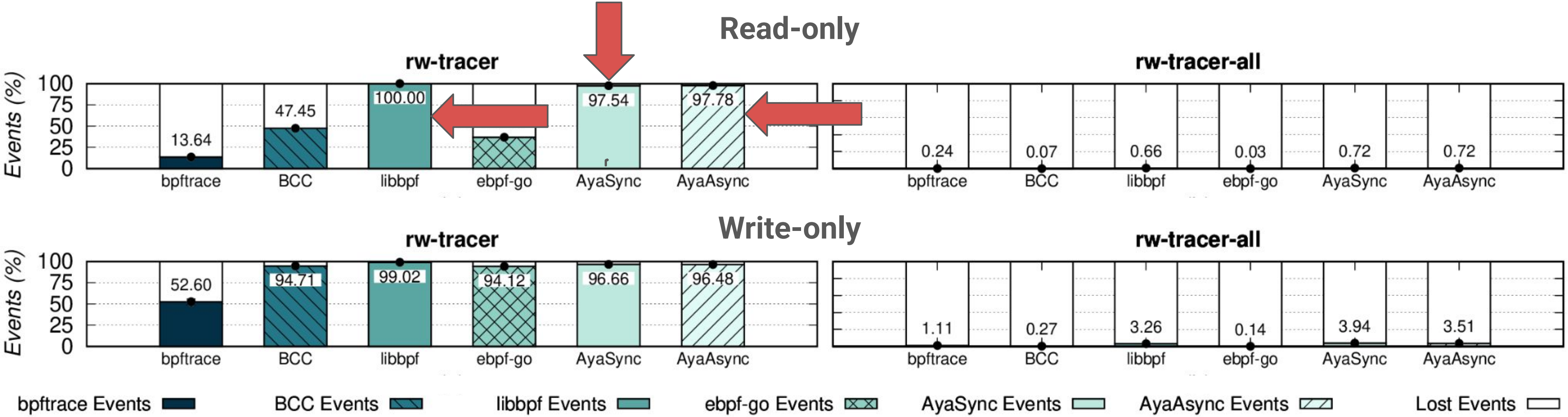
- Also influenced by the amount of data sent to user space
    - While rw-tracer captures most of the events, rw-tracer-all loses most of them

Experimental Results - Fidelity



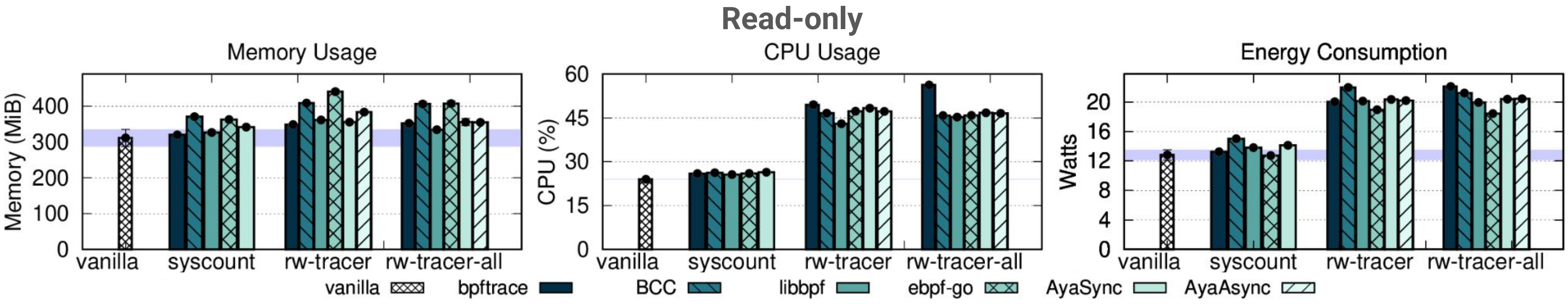
- Impacted by the workload event generation rate
    - Faster workloads like read-only cause more event loss overall
- Also influenced by the amount of data sent to user space
    - While rw-tracer captures most of the events, rw-tracer-all loses most of them
- Aya and libbpf capture most events, while bpftrace struggles considerably

Experimental Results - Fidelity

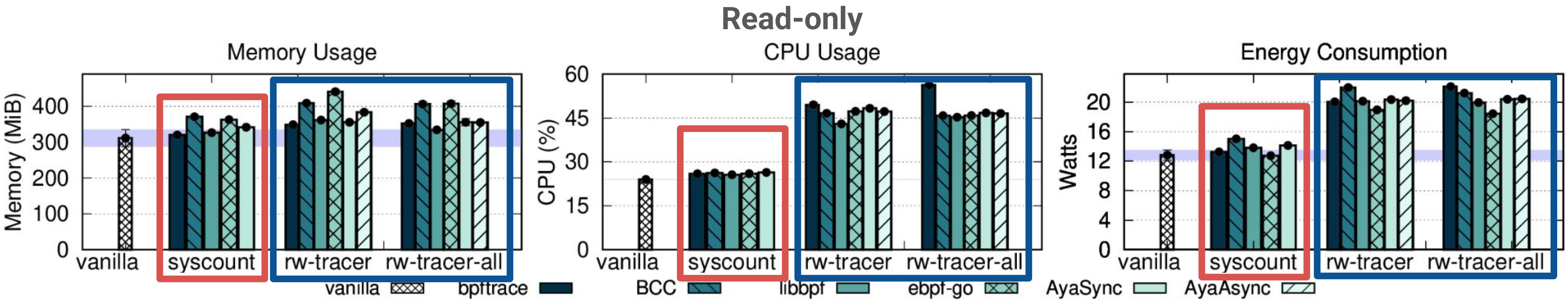


- Impacted by the workload event generation rate
    - Faster workloads like read-only cause more event loss overall
- Also influenced by the amount of data sent to user space
    - While rw-tracer captures most of the events, rw-tracer-all loses most of them
- Aya and libbpf capture most events, while bpftrace struggles considerably

Experimental Results - Resource Usage

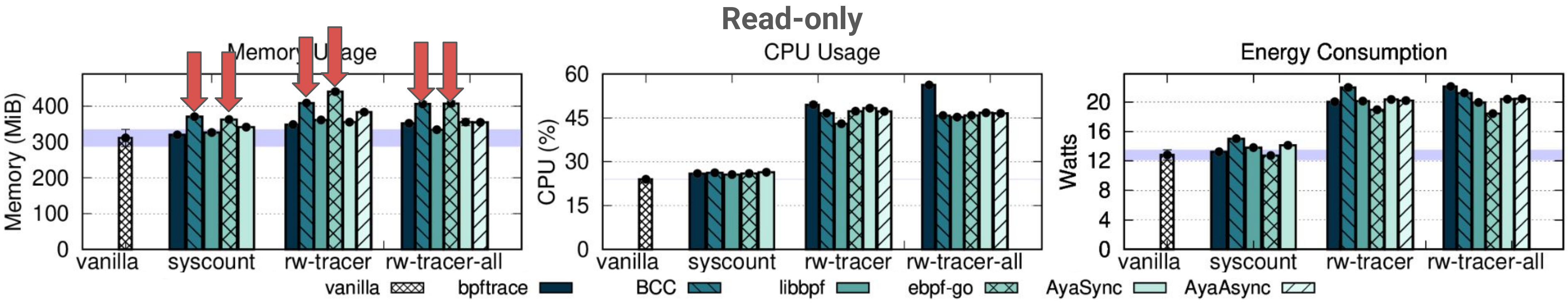


Experimental Results - Resource Usage



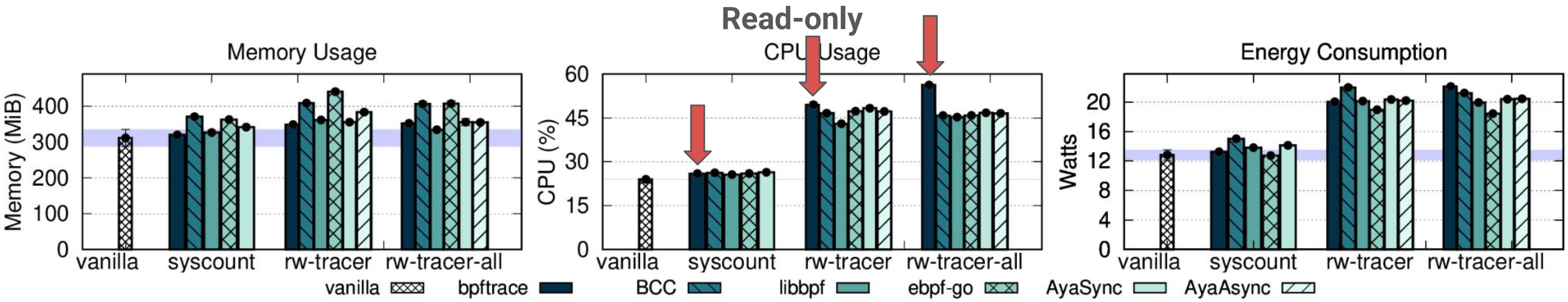
- Higher eBPF tool complexity
  - higher resource usage

Experimental Results - Resource Usage



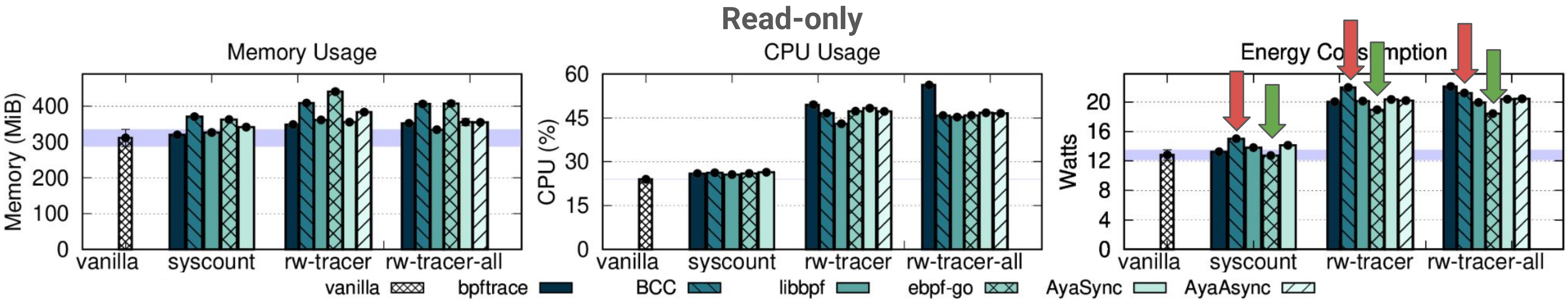
- Higher eBPF tool complexity
  - higher resource usage
- Memory: BCC and ebpf-go as the most demanding

Experimental Results - Resource Usage



- Higher eBPF tool complexity
    - higher resource usage
- Memory: BCC and ebpf-go as the most demanding
- CPU: bpftrace as most demanding

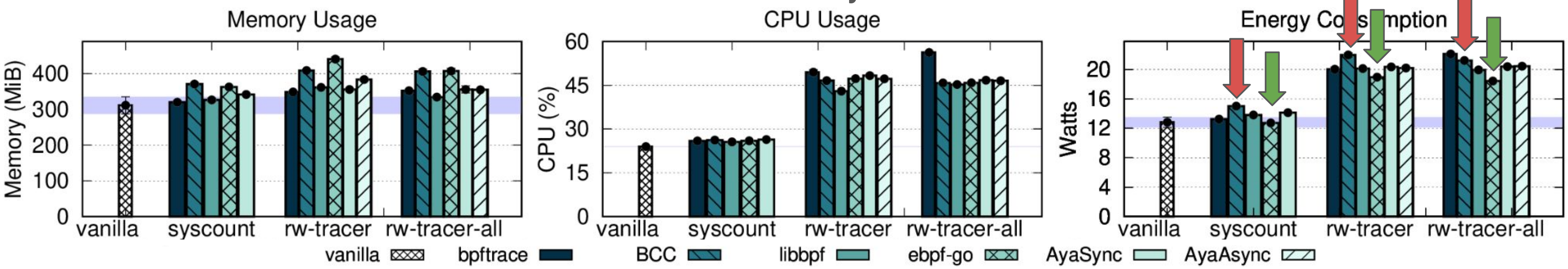
Experimental Results - Resource Usage



- Higher eBPF tool complexity
    - higher resource usage
- Memory: BCC and ebpf-go as the most demanding
- CPU: bpftrace as most demanding
- Energy: BCC among the most demanding and ebpf-go as more efficient

Experimental Results - Resource Usage

Read-only

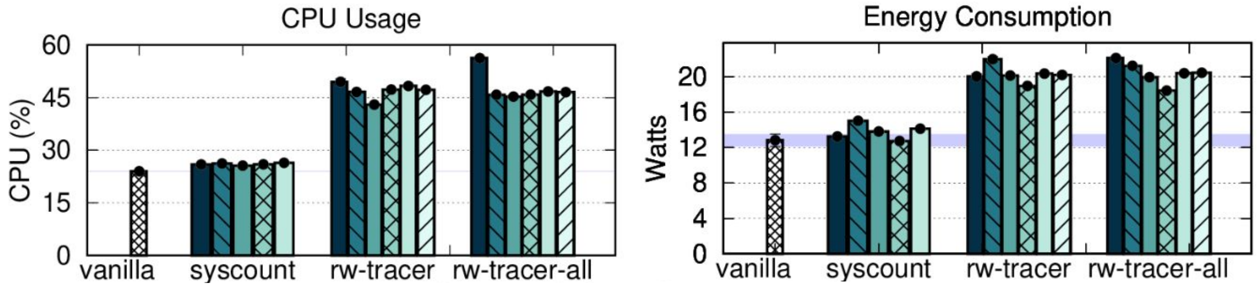


- Higher eBPF tool complexity
  - higher resource usage
- Memory: BCC and ebpf-go as the most demanding
- CPU: bpftrace as most demanding
- Energy: BCC among the most demanding and ebpf-go as more efficient

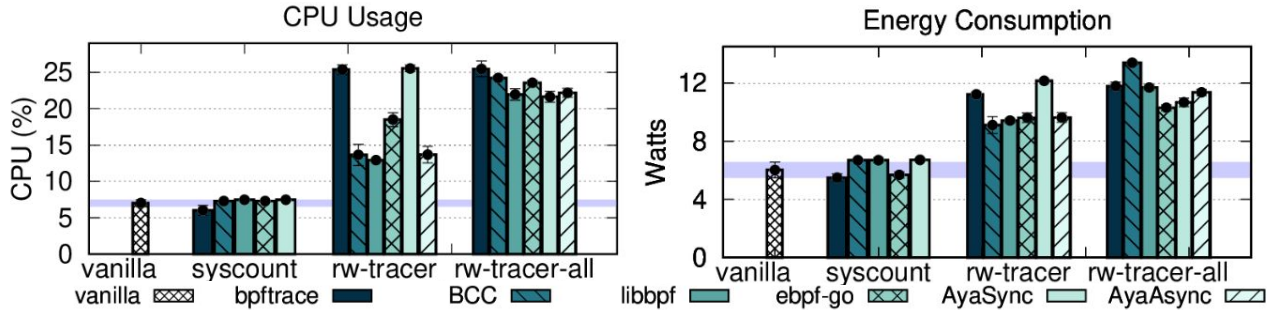
Conclusions become **less clear** for **less intensive workloads**!

Experimental Results - Resource Usage

Read-only

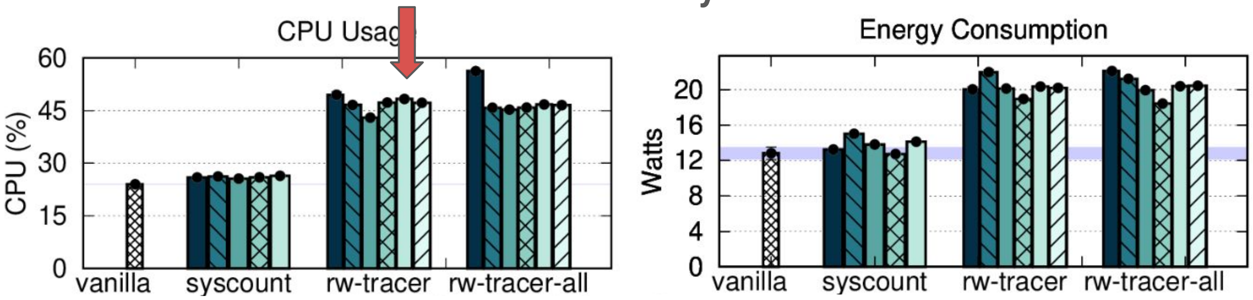


Write-only

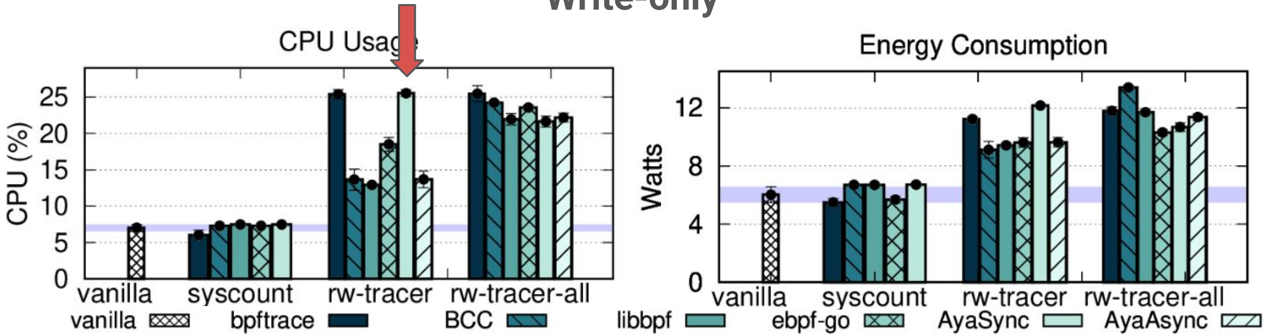


Experimental Results - Resource Usage

Read-only

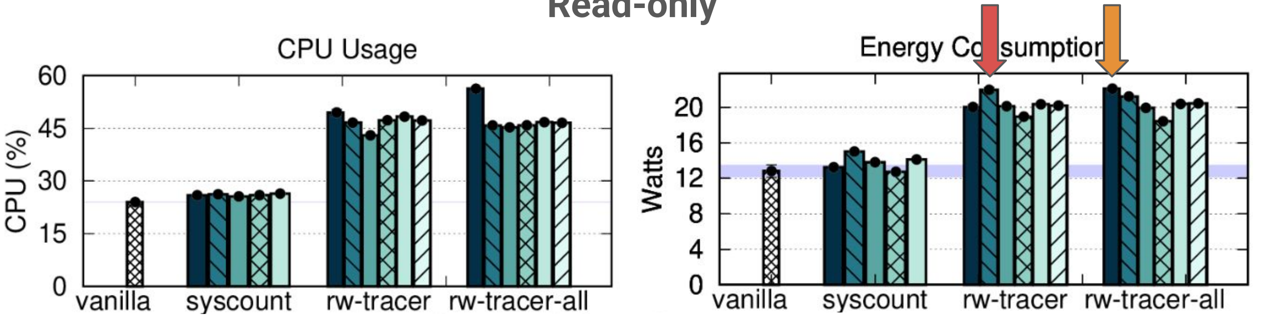


Write-only

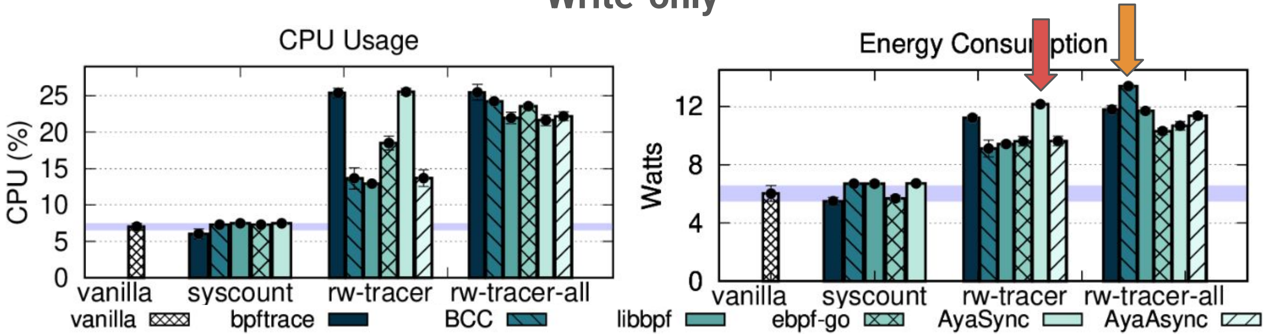


Experimental Results - Resource Usage

Read-only



Write-only



Key Takeaways

- Fidelity vs Performance
  - Trade-off driven by workload intensity and event size
- Programming Language Impact
  - High-level abstractions are convenient but costly in performance/resources
- Polling Strategy vs Resource Usage
  - Active Polling (AyaSync): higher CPU and energy usage
  - Epoll-based strategies make it more efficient

Key Takeaways

- Fidelity vs Performance
  - Trade-off driven by workload intensity and event size
- Programming Language Impact
  - High-level abstractions are convenient but costly in performance/resources
- Polling Strategy vs Resource Usage
  - Active Polling (AyaSync): higher CPU and energy usage
  - Epoll-based strategies make it more efficient

eBPF libraries **behave differently** under varying conditions → **deeper quantitative assessment** is needed**!**

Future Work

Configurations and deployment

- Vary eBPF configurations (e.g. ring buffer size, polling timeout)
- Isolate Kernel and user space components

Future directions

- Expand to other domains (e.g. network, security)
- Assess performance under real-world workloads
- Evaluate complex eBPF applications

3rd Workshop on eBPF and Kernel Extensions (eBPF)

No Two Snowflakes Are Alike: Studying eBPF Libraries’ Performance, Fidelity and Resource Usage

dsrhaslab/ebpf-lib-eval

carlos.e.machado@inesctec.pt