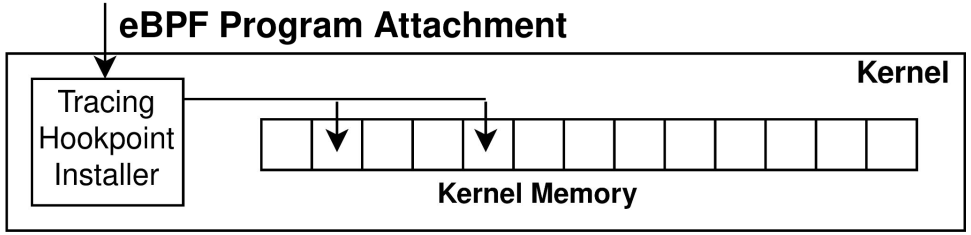
Eliminating eBPF Tracing Overhead on Untraced Processes

Milo Craun, Khizar Hussain, Uddhav Gautam, Zhengjie Ji, Tanuj Rao, and Dan Williams

eBPF For Tracing

- eBPF used for dynamic system tracing and observability
- Attach to *Tracing Hookpoints*
  - tracepoints and kprobes
- *Tracing Hookpoint Installer* changes kernel text pages to install the program
  - patching no-op for tracepoint
  - installing trap/interrupt instruction for kprobe

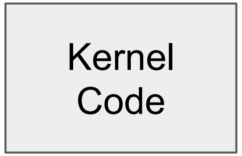


Per-process Tracing with eBPF

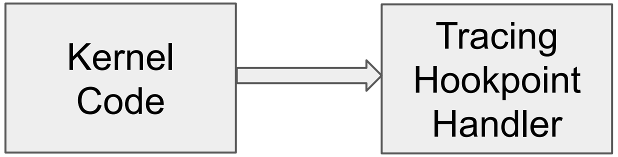
- We find current tracing is coarse grained
    - once activated, hookpoint triggers for all processes
- Need for per-process tracing
    - Want to trace a single application or a set of applications
    - Not natively supported

Approaches to Per-process

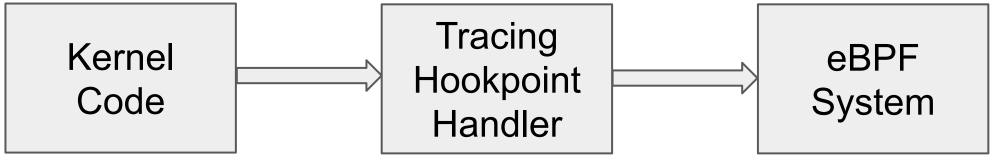
Approaches to Per-process



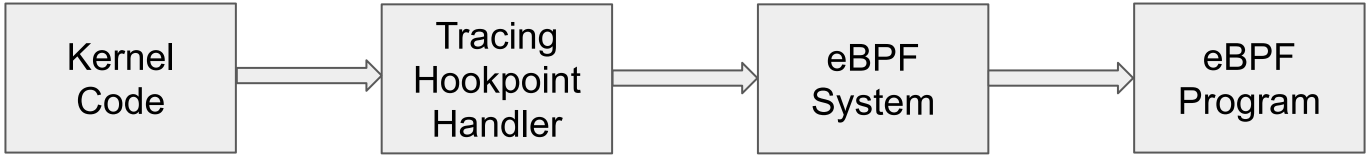
Approaches to Per-process



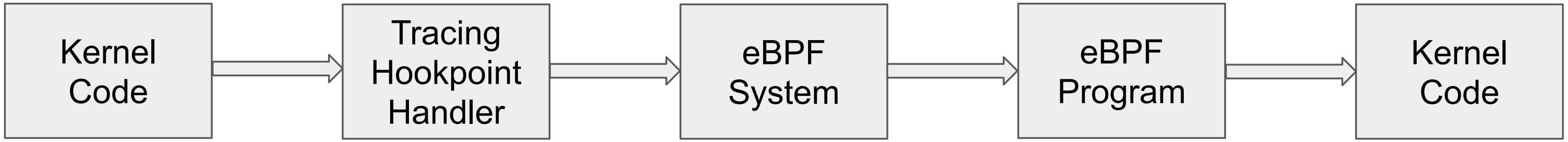
Approaches to Per-process



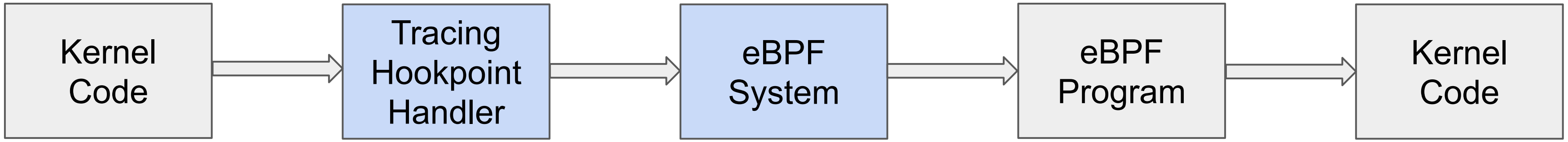
Approaches to Per-process



Approaches to Per-process



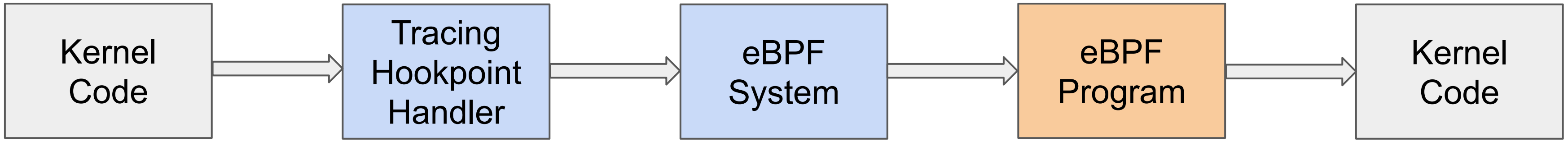
Approaches to Per-process



Pre-eBPF

- Filter for PID before BPF program
- Not supported by kernel

Approaches to Per-process



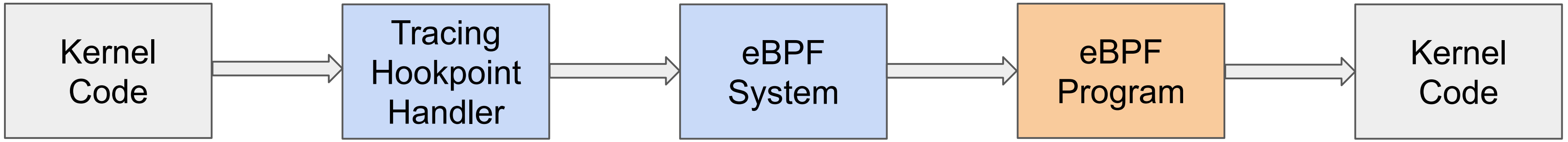
Pre-eBPF

- Filter for PID before BPF program
- Not supported by kernel

In-eBPF

- Filter for PID in BPF program
- Used by bpftrace

Approaches to Per-process



Pre-eBPF

- Filter for PID before BPF program
- Not supported by kernel

In-eBPF

- Filter for PID in BPF program
- Used by bpftrace

Post-eBPF

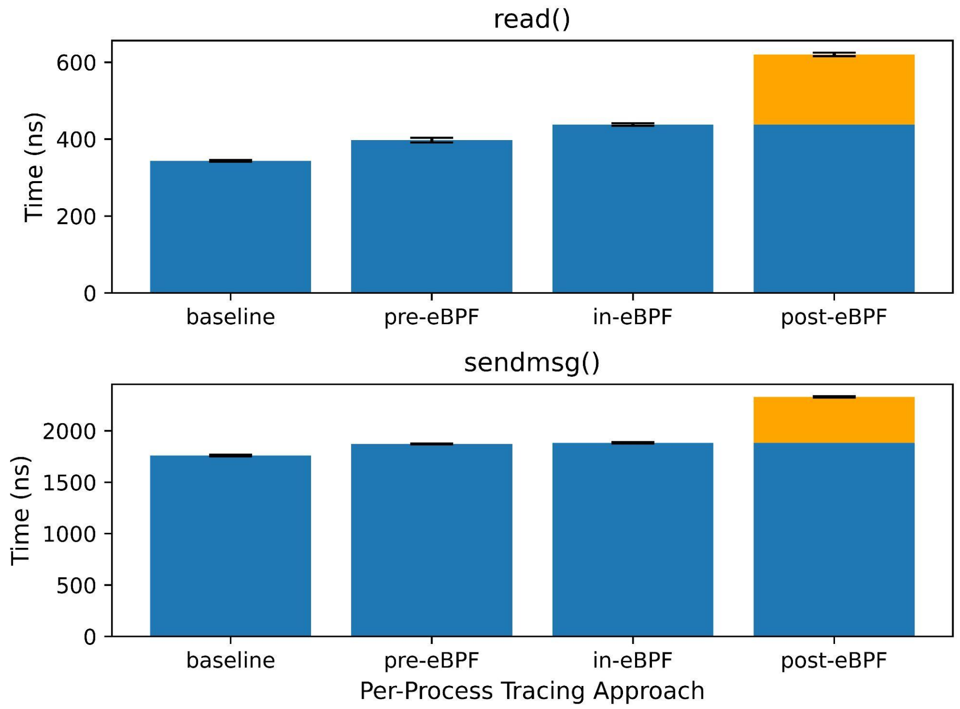
- Filter for PID after data collection
- Intuitively makes sense

Overheads of Per-process Tracing

- Filtering approaches cause *untraced overhead*
  - overhead on processes that are not traced
- Performed experiments to measure the overhead

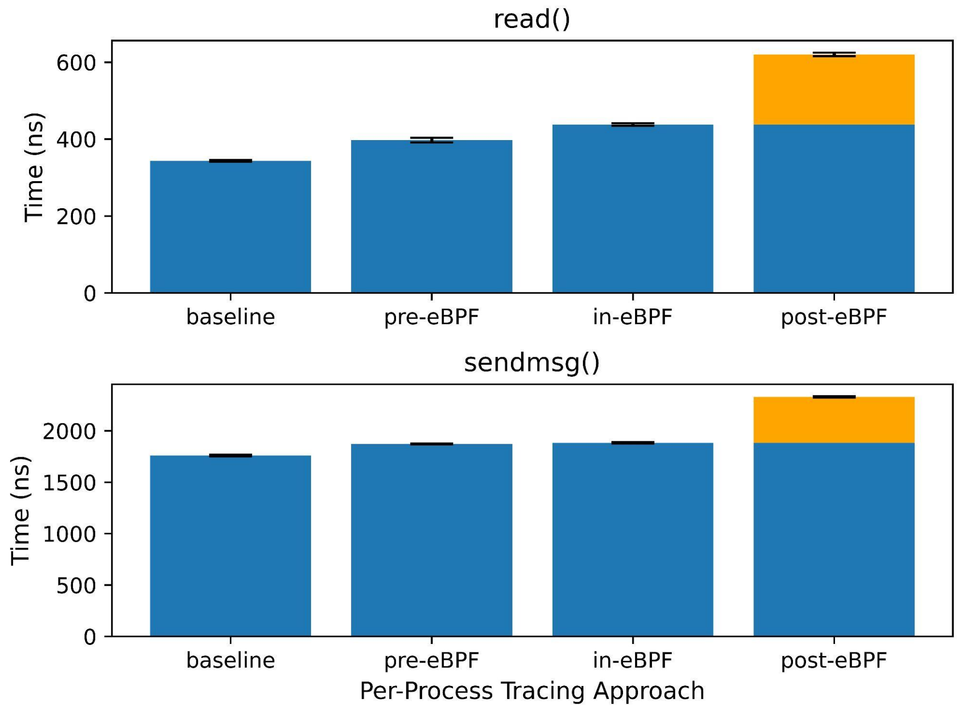
Overheads of Per-process Tracing

- Filtering approaches cause *untraced overhead*
  - overhead on processes that are not traced
- Performed experiments to measure the overhead



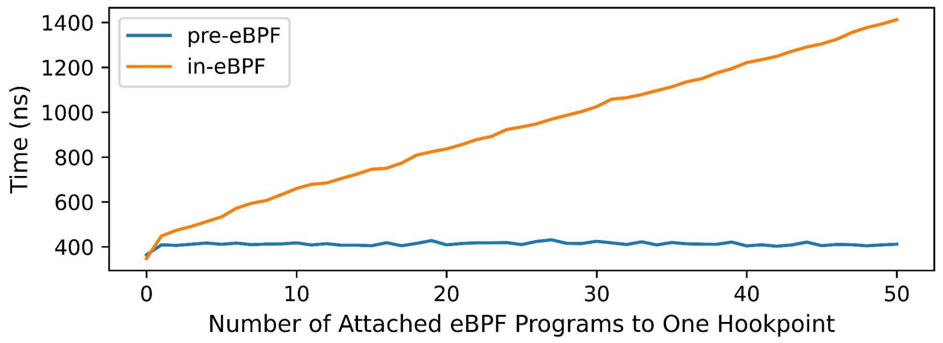
Overheads of Per-process Tracing

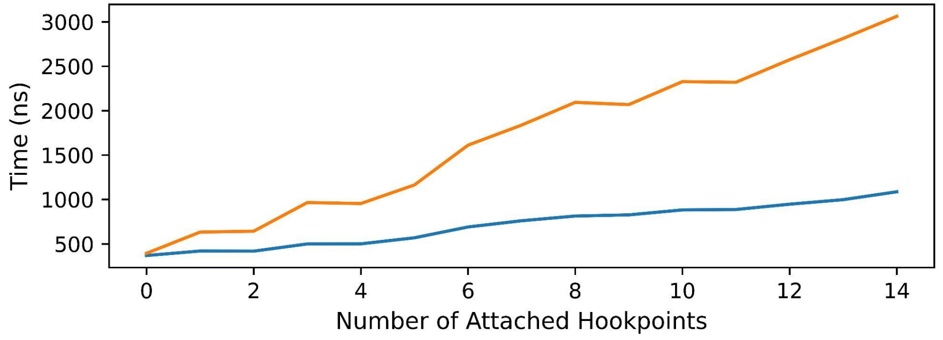
- Filtering approaches cause *untraced overhead*
  - overhead on processes that are not traced
- Performed experiments to measure the overhead
- Memcached Throughput
  - pre-eBPF: 1.5% decrease
  - in-eBPF: 2.7% decrease



Overheads of Per-process Tracing

- Filtering approaches cause *untraced overhead*
  - overhead on processes that are not traced
- Performed experiments to measure the overhead
- Memcached Throughput
  - pre-eBPF: 1.5% decrease
  - in-eBPF: 2.7% decrease
- Trends seem to indicate more eBPF programs attached





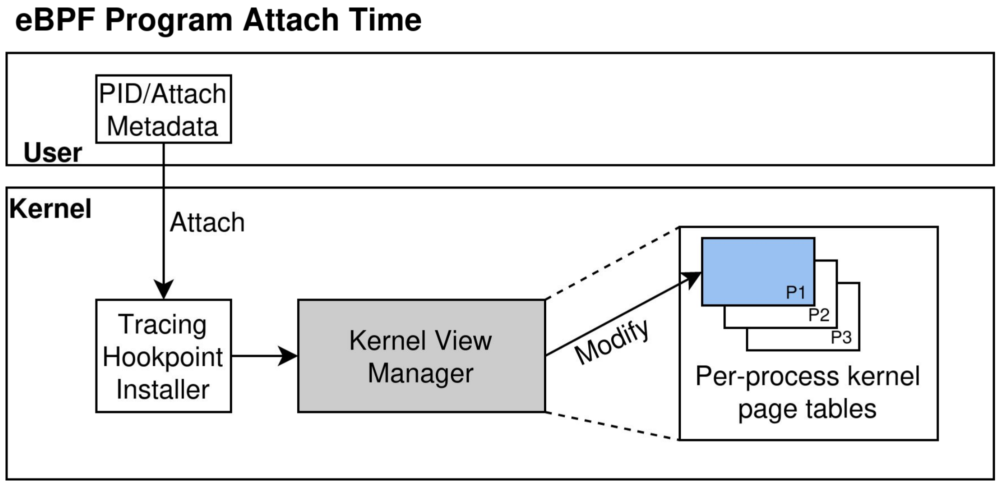
How can we achieve per-process tracing without untraced overhead?

Key Insights

- Unattached tracing hookpoints are fast
  - tracepoints are optimized
  - kprobes are not installed
- Attaching a tracing program requires writes to kernel text pages

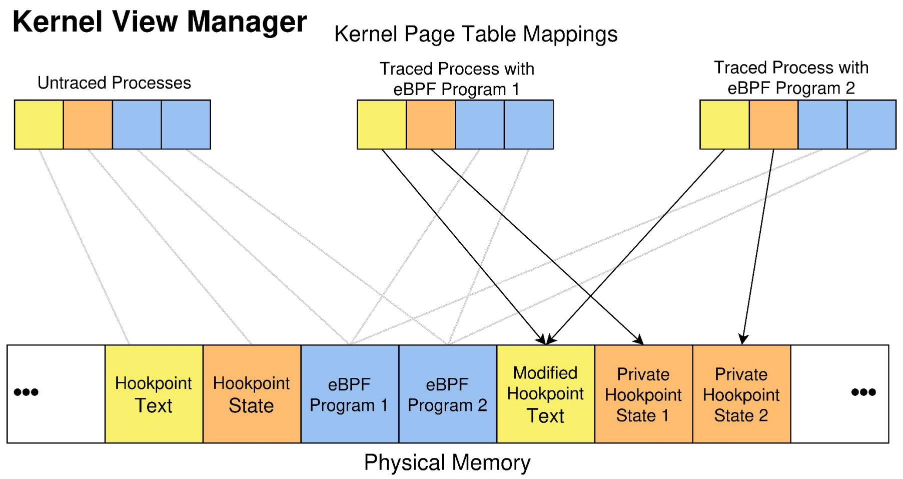
Per-process Kernel Views

- Give each process its own view of the kernel
- Allows for per-process sets of kernel hookpoints
- Modify tracing hookpoint installers to communicate with a *kernel view manager*



Kernel View Manager

- Creates copies of kernel text pages
- Modifies process kernel page tables to map copies of pages
- Installs tracing program
- Provides private hookpoint state


Kernel View Manager
Kernel Page Table Mappings
Untraced Processes
Traced Process with eBPF Program 1
Traced Process with eBPF Program 2
•••
Hookpoint Text
Hookpoint State
eBPF Program 1
eBPF Program 2
Modified Hookpoint Text
Private Hookpoint State 1
Private Hookpoint State 2
•••
Physical Memory

Open Questions

1. Tracing Hookpoint State
    a. What does this consist of? How can we manage it?

Open Questions

1. Tracing Hookpoint State
    a. What does this consist of? How can we manage it?
2. Issues with virtual mappings
    a. What does fork() do? Implications for other subsystems?

Open Questions

1. Tracing Hookpoint State
    a. What does this consist of? How can we manage it?
2. Issues with virtual mappings
    a. What does fork() do? Implications for other subsystems?
3. Wasted memory
    a. How much waste does creating copies of kernel pages incur?

Open Questions

1. Tracing Hookpoint State
    a. What does this consist of? How can we manage it?
2. Issues with virtual mappings
    a. What does fork() do? Implications for other subsystems?
3. Wasted memory
    a. How much waste does creating copies of kernel pages incur?
4. Overhead of multiple kernel views
    a. What are the costs associated with changing page tables?

Extending Kernel Views

1. Different Granularities
    a. Other extension use cases need different granularity
    b. Per-flow kernel views
    c. Need to be able to identify and manage different network flows

Extending Kernel Views

1. Different Granularities
   a. Other extension use cases need different granularity
   b. Per-flow kernel views
   c. Need to be able to identify and manage different network flows
2. Increased Kernel State Management Complexity
   a. Ensure right kernel view is loaded when events occur
   b. Change kernel view for interrupts
   c. Private state for eBPF programs

Takeaways

- Identified that existing approaches for per-process tracing impose overheads on untraced processes
- We propose a system that uses virtual memory mappings to eliminate the overhead on untraced processes
- A kernel view manager gives each process its own view of the kernel

Questions?