

Build ID + Offset Sampling in Linux Perf

Ian Rogers, Google

Tracing Microconference



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Overview

01

Background

Virtual address sampling, procfs scanning, and symbol resolution in Linux perf.

02

Build ID Motivation & Implementation

Eliminating mmap event streams, preventing ASLR leaks, and kernel/user space changes.

03

Overhead Analysis

Evaluating memory footprint, sample size trade-offs, and profiling performance.

04

Proposals to Reduce Memory Size

Lazy injection, BuildID filtering, stack compression, and hybrid sampling models.



Virtual Address Sampling

01

Virtual Memory Address Capture

Traditional perf event sampling captures virtual memory addresses (IPs) directly in performance event samples.

02

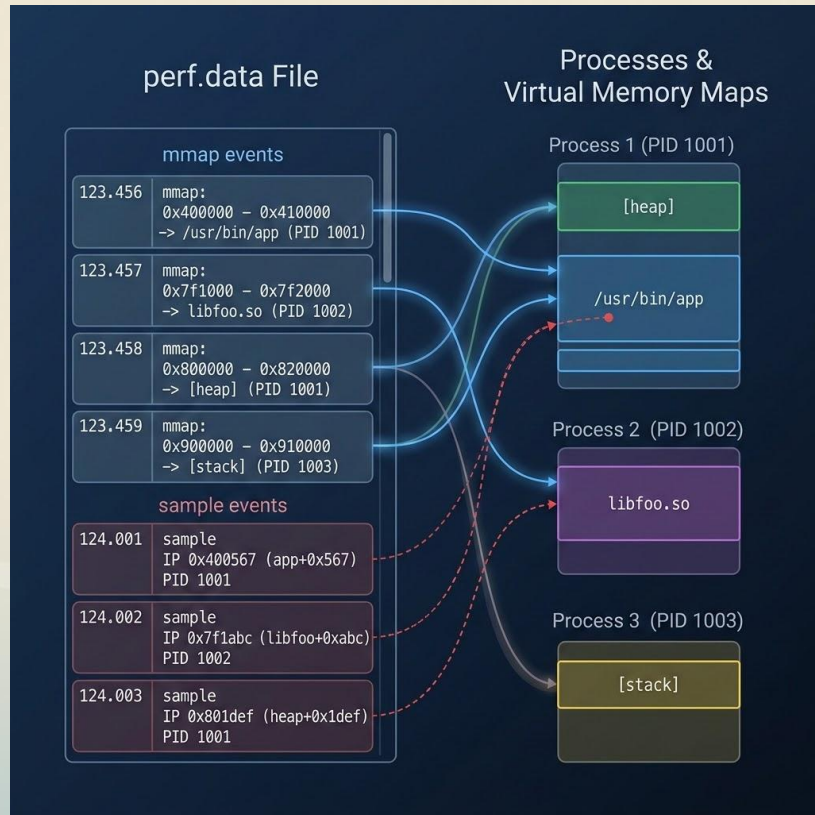
Symbol Resolution

Symbols resolved by cross-referencing virtual addresses with mmap metadata.

03

Mmap Event Dependency

Heavy reliance on synthesizing and maintaining mmap event streams throughout execution.



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Virtual Address Sampling

01

Virtual Memory Address Capture

Traditional perf event sampling captures virtual memory addresses (IPs) directly in performance event samples.

02

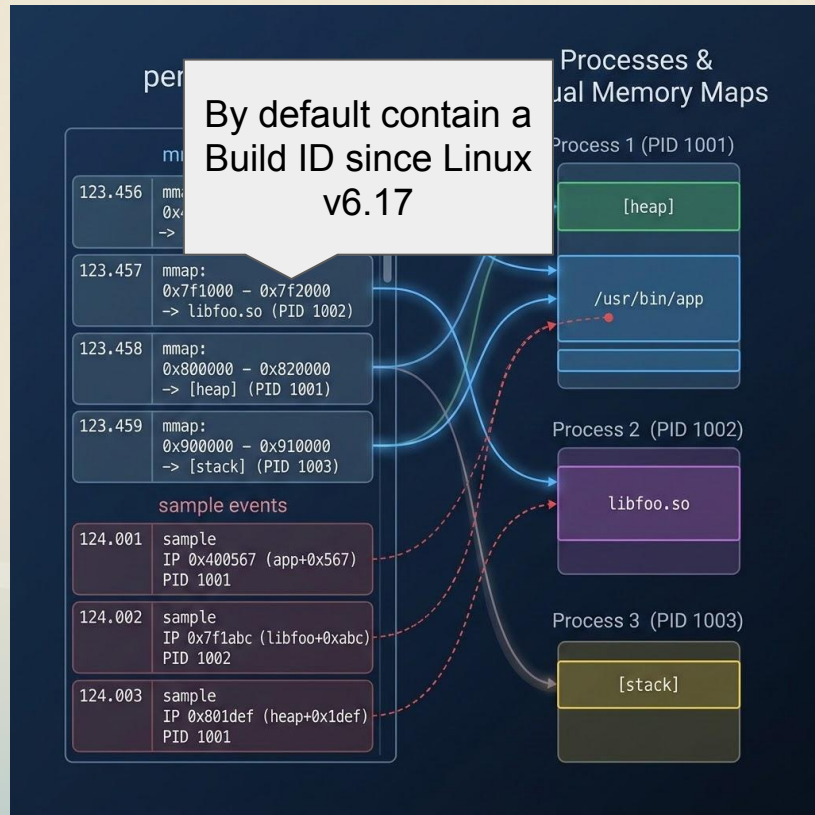
Symbol Resolution

Symbols resolved by cross-referencing virtual addresses with mmap metadata.

03

Mmap Event Dependency

Heavy reliance on synthesizing and maintaining mmap event streams throughout execution.



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Motivation

Scalability Bottleneck:

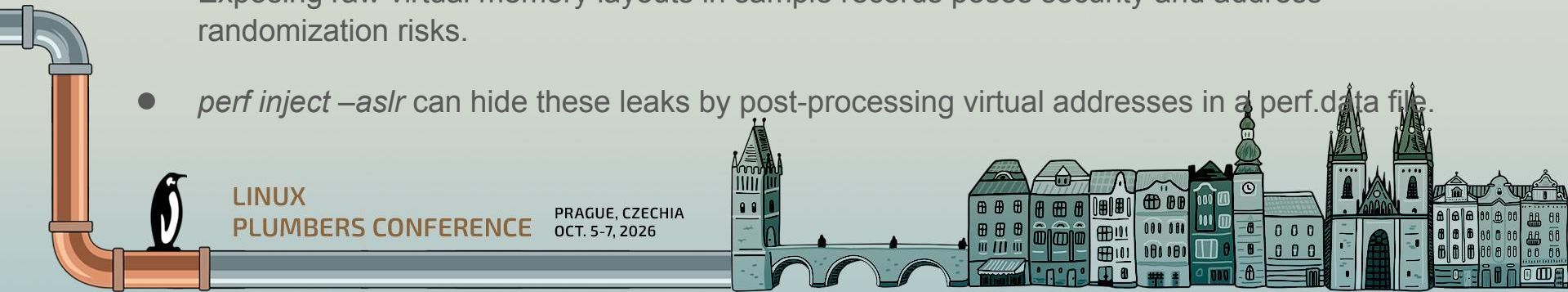
- Scanning procs at startup generates a massive mmap event stream.

Storage Footprint:

- Mmap metadata dominates sample data for short runs or samples on infrequent events.

ASLR Address Leaks:

- Exposing raw virtual memory layouts in sample records poses security and address randomization risks.
- *perf inject -aslr* can hide these leaks by post-processing virtual addresses in a perf.data file.



BPF Build ID stack traces

- **ELF Header Inspection**

Reads the Build ID directly from the ELF note segment in the binary's first memory page during stack unwinding.

- **Page Cache Resident Dependency**

Requires the header page in page cache; BPF unwinding may run in NMI/atomic context and cannot handle page faults.

- **Mitigating Misses via mlock**

If non-resident, Build ID retrieval fails. Profilers pin (mlock) header pages to guarantee uninterrupted resolution.



RFC v1 data layout

New Perf Sample Flags

As well as flags to have virtual addresses in a sample, introduce new flags that add Build ID and Offset:

- `PERF_SAMPLE_IP -> PERF_SAMPLE_BUILD_ID_OFFSET`
- `PERF_SAMPLE_CALLCHAIN -> PERF_SAMPLE_CALLCHAIN_BUILD_ID_OFFSET`

The Build ID is 24 bytes followed by an 8 byte offset:

```
/*  
 * Build IDs may be present in a number of events. They have a  
 * consistent encoding of:  
 *  
 * struct build_id {  
 *     u8          size;  
 *     u8          __reserved_1;  
 *     u16         __reserved_2;  
 *     u8          data[20];  
 * };  
 */  
char data[20]; } && PERF_SAMPLE_BUILD_ID_OFFSET  
{ struct build_id bid;  
  u64 offset; } && PERF_SAMPLE_BUILD_ID_OFFSET  
{ u64 nr;  
  struct {  
      struct build_id bid;  
      u64 offset;  
  }[nr]; } && PERF_SAMPLE_CALLCHAIN_BUILD_ID_OFFSET
```



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



RFC v2 data layout

Extend the v1 data layout.

Add deferred sample cookies so that a kernel stack trace can pair with a user stack trace.

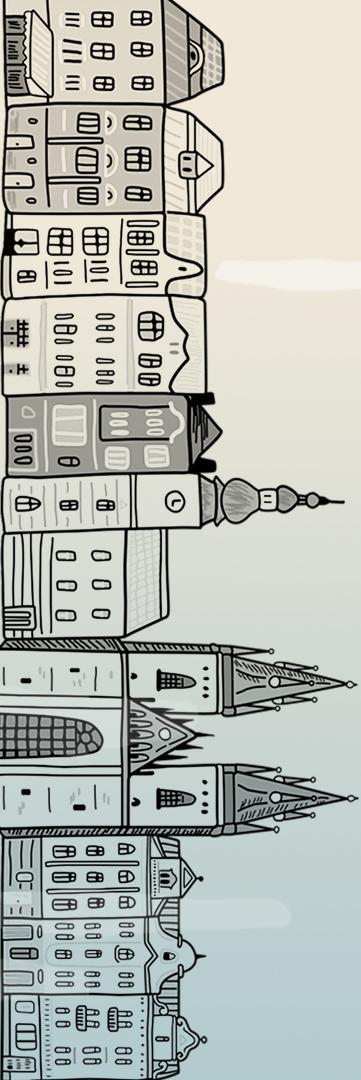
Deferred samples use misc bit to indicate Build ID + offset encoding.

Savings may happen if >1 kernel stack trace per user stack trace.

User stack traces occur at faultable locations.

```
* struct {  
*     struct perf_event_header    header;  
*     u64                        cookie;  
*     u64                        nr;  
*     union {  
*         u64                    ips[nr];  
*         struct {  
*             struct perf_build_id bid;  
*             u64                    offset;  
*         }[nr]; && (header.misc & PERF_RECORD_MISC_MMAP_BUILD_ID)  
*     };  
*     struct sample_id            sample_id;  
* };  
*/  
PERF_RECORD_CALLCHAIN_DEFERRED    = 22,
```



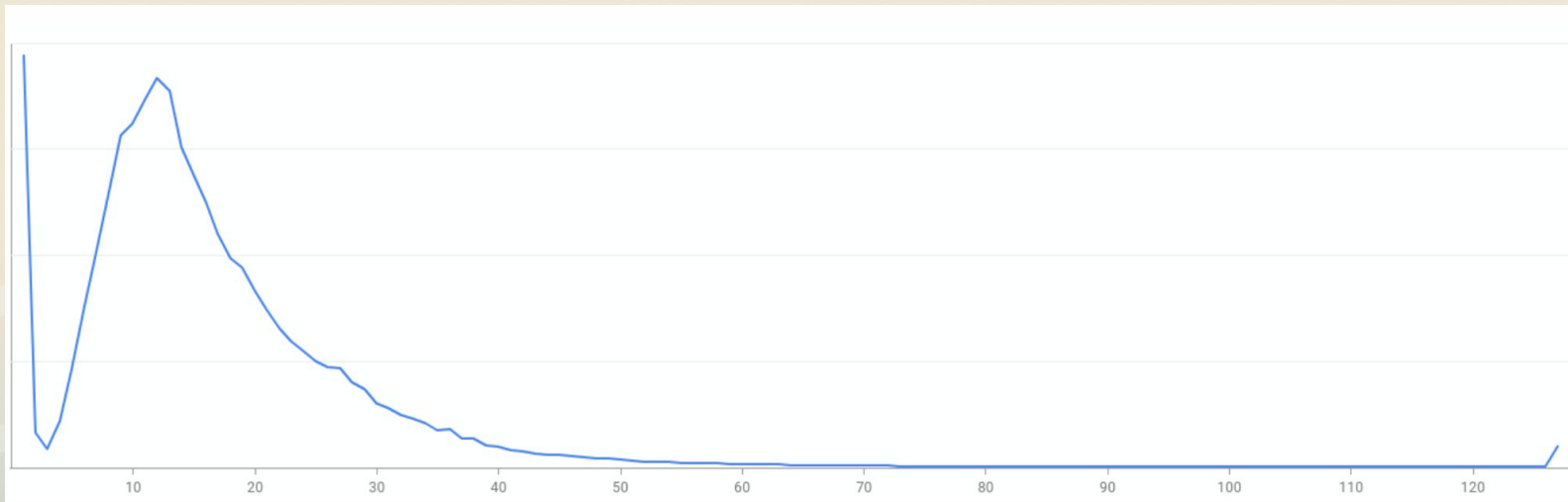


Size Trade-Off

- **Inline Sample Overhead:** Capturing Build ID + offset adds inline bytes (32 bytes vs 8 bytes per address).
- **Callchain Impact:** Stack frames increase by ~24 bytes per callchain entry when embedding Build IDs.
- **Metadata Savings:** Eliminating global MMAP2 events (72+ bytes each) across thousands of idle/unsampled tasks.



Average stack depth for Google Wide Profiling cycles



Peaks at 1 (unwinding failed), 12 and 127 (max unwind)



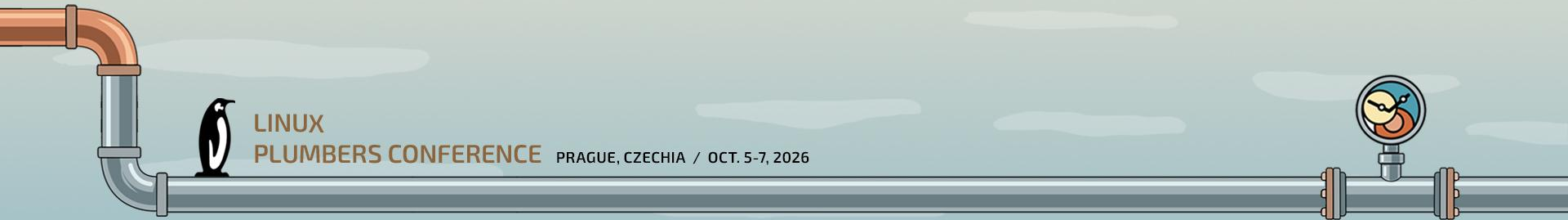
**LINUX
PLUMBERS CONFERENCE**

PRAGUE, CZECHIA / OCT. 5-7, 2026



Analysis on GWP cycles

- Typical SAMPLE:MMAP ratio for a 10 second GWP = 45:1
- For a typical 23MB of SAMPLE_IP events expect a **2MB size increase**.
- Assuming 20 callchain entries per sample, expect a **3.1x size increase**.



Analysis on GWP cycles

- Typical SAMPLE:MMAP ratio for a 10 second GWP = 45:1
- For a typical 23MB of SAMPLE_IP events expect a 23MB size increase.
- Assuming 20 callchain entries per sample expect a 460MB size increase.

Assumes a high ratio of
samples to mmap events



Saving Memory With Hybrid Sampling

01 | DYNAMIC STRATEGY

Selective Combination

Open the event both for virtual address (disabled) and Build ID + offset sampling (enabled).

Build ID + Offset

+

Virtual Address (VA)

02 | PROCESS DETECTION

Hot Process Identification

Samples occur in hot processes, synthesize mmap events for these processes. Use BPF to select virtual address samples for processes with mmap data.

1. Build ID Sampling



2. Active Binaries Detected



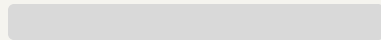
3. Synthesized mmap

03 | OPTIMIZATION

Minimal Footprint Target

By sampling and synthesizing the number of mmap events is reduced to that of just hot processes. The large build ID + offset samples are used just for cold processes.

Standard Overhead

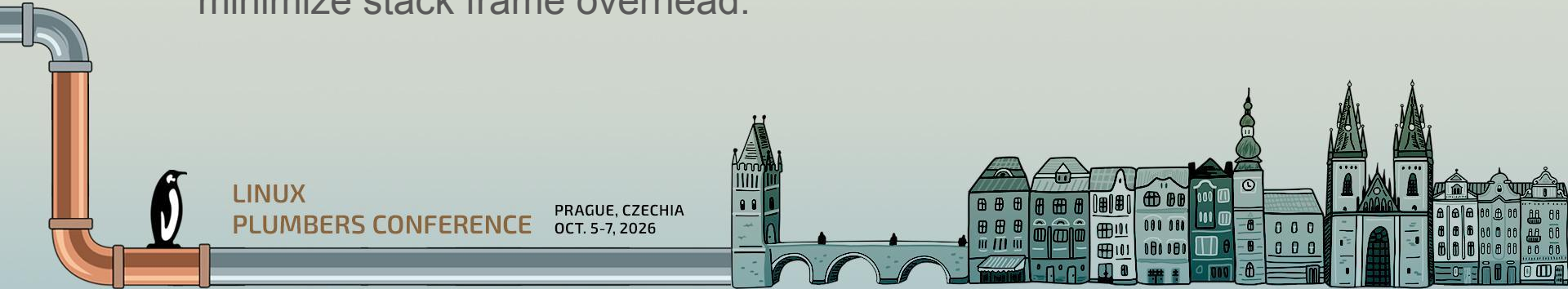


Hybrid Scheme (perf.data)

MINIMAL SIZE

Callchain Compression Techniques

- **Redundancy Exploitation:** High redundancy of Build IDs across stack frames in deep callchains.
- **Lookup Tables:** Storing Build IDs in a sample-local or record-wide lookup table.
- **Encoding Schemes:** Utilizing run-length encoding and compression to minimize stack frame overhead.



Open Problems

Data sampling

Stack and heap don't have build ids.

Code without Build IDs

JIT and other such code don't have valid build ids.

Virtual addresses in auxiliary and other kinds of event

Need to expand every kind of event for build ids otherwise mmap synthesis is still necessary.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Conclusions

- Build ID + offset samples avoid startup overhead and ASLR memory leaks, but it comes with a potential size regression.
- Hybrid sampling or compression may mean that the approach is always a net size win over existing virtual addresses.
- Various issues exist beyond memory overhead.
- **Respond to the RFC or let's talk if you have feedback!**

