LTTng and Related Projects Update

DORSAL Progress Report Meeting

February 2026

EfficiOS

Who is EfficiOS?

Our work
these days

What is LTTng?
What is Babeltrace?

What is LTTng?

Extremely low overhead troubleshooting tool

- First released in 2005
- Open Source

A collection of projects

- Kernel tracer (LTTng-modules)
- User space tracer (LTTng-UST)
- Tracing control tools (LTTng-tools)

What is Babeltrace 2?

Trace manipulation toolkit

- Allows decoding and transformation of trace files
- Extensible via a plug-in architecture
  - Support other trace formats
  - Implement analyses

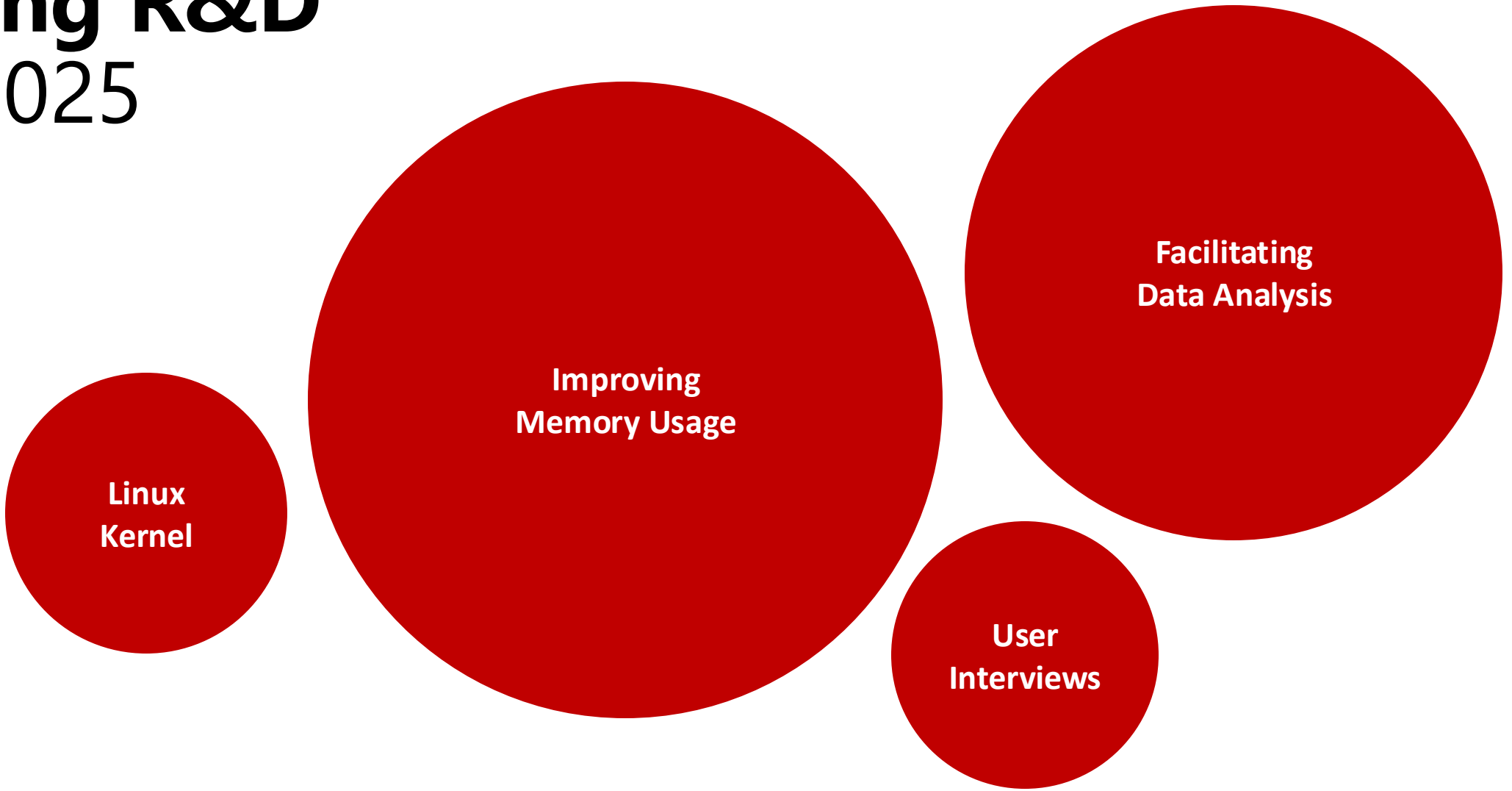
Update Outline

LTTng Research & Development

- In 2025
- Ongoing

2025 Research & Development

LTTng R&D in 2025



Tackling our memory footprint

Shared buffers

A detour to the ring buffers!

Buffer stall recovery

Feature: Recover from per-user ring buffer stalled state

Advantages

- Gracefully recover when an application is asynchronously terminated while writing to the ring buffer
- Warn when an application is stopped for a long time while writing to the ring buffer

Disadvantage

- Additional overhead when writing an event to the ring buffer

Tackling our memory footprint

Shared buffers

Feature: Per-channel buffers

Advantages

- Memory footprint does not scale with number of CPUs
- More resistant to variability in trace production
- More intuitive traces

Disadvantage

- More contention when writing trace data to buffers

Allocating buffers on-demand

Feature: **Preallocation policy**

Advantages

- Lower initial memory footprint
- Widens viability of per-CPU buffers

Disadvantages

- Small latency when need to allocate a subbuffer
- Less predictable memory usage

Reclaim unused memory

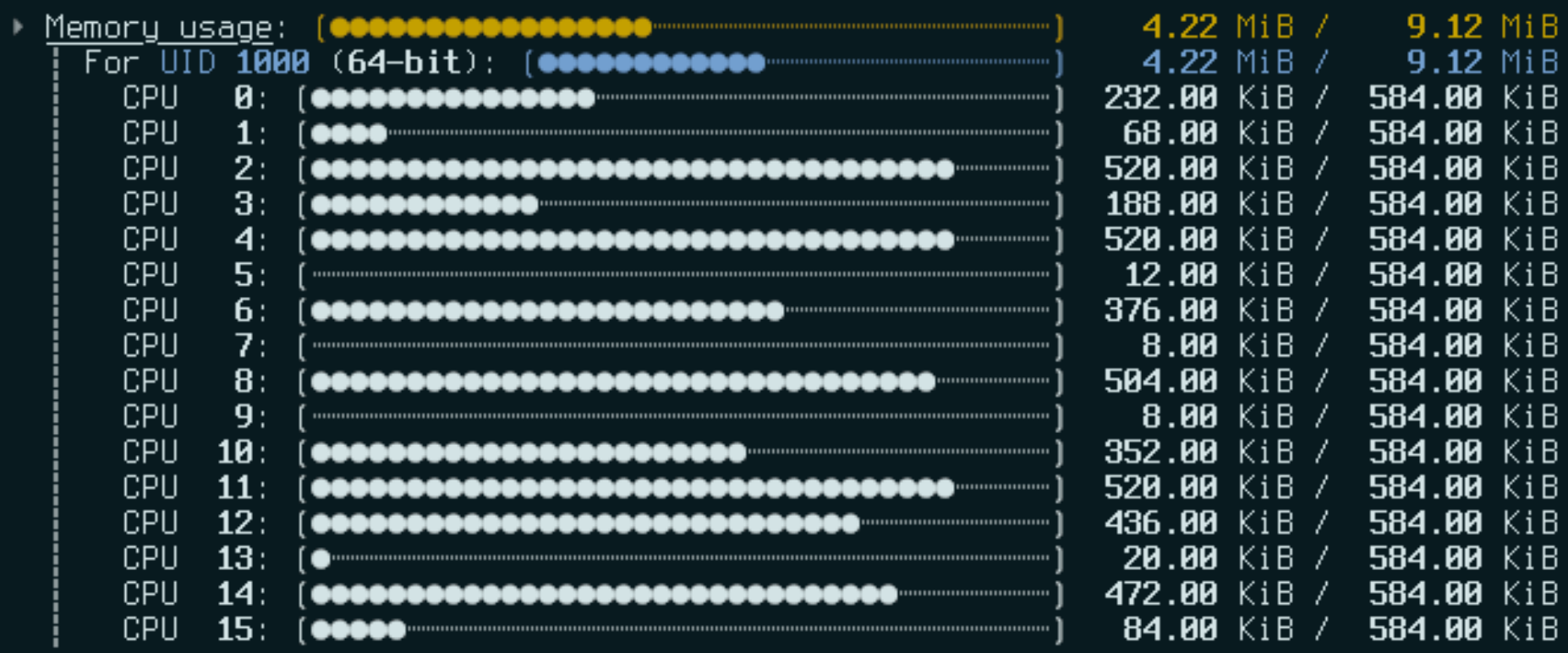
Feature: Reclaim memory command

Advantages

- Scale memory usage based on tracing load

Disadvantage

- Some additional overhead
- Small latency when need to (re)allocate a subbuffer


Memory usage: 4.22 MiB / 9.12 MiB
For UID 1000 (64-bit): 4.22 MiB / 9.12 MiB
CPU 0: 232.00 KiB / 584.00 KiB
CPU 1: 68.00 KiB / 584.00 KiB
CPU 2: 520.00 KiB / 584.00 KiB
CPU 3: 188.00 KiB / 584.00 KiB
CPU 4: 520.00 KiB / 584.00 KiB
CPU 5: 12.00 KiB / 584.00 KiB
CPU 6: 376.00 KiB / 584.00 KiB
CPU 7: 8.00 KiB / 584.00 KiB
CPU 8: 504.00 KiB / 584.00 KiB
CPU 9: 8.00 KiB / 584.00 KiB
CPU 10: 352.00 KiB / 584.00 KiB
CPU 11: 520.00 KiB / 584.00 KiB
CPU 12: 436.00 KiB / 584.00 KiB
CPU 13: 20.00 KiB / 584.00 KiB
CPU 14: 472.00 KiB / 584.00 KiB
CPU 15: 84.00 KiB / 584.00 KiB

Towards facilitating data analysis

Moving towards CTF 2

Babeltrace 2.1 – Add reading and producing CTF 2 traces

- Release: Q1 2025

LTTng 2.15 – Add producing CTF 2 traces

- **Release candidate currently published** for testing
- Release: Q1 2026

Benefits of CTF 2

- Better trace readability
- Broader type support
- Cleaner trace streaming
- Easier decoding

Upcoming Work

Lightweight insight

Feature: Aggregation maps, trace hit counters

- Count number of times an event (or event set) occurs
- Super lightweight – Doesn't require tracing buffers
- Powerful building block for responsive trace control
- Upcoming: LTTng 2.16

Ongoing R&D

LTTng R&D Objectives

- Reduce **overhead**
- Improve **instrumentation**
- Improve trace data **relevance**
- Facilitate tracing **configuration**
- Enhance **trace model** for:
  - Trace presentation,
  - Analysis automation.

Lightweight Event Content Statistics

- Objectives: [**overhead**]
- Lightweight event content statistics without tracing
- In-place aggregation of statistics distributions with less overhead than the LTTng ring buffer
- Extension of LTTng trace hit counters
- Distribution bucket indexes are a function of event field payload values
- These statistics distributions can then be visualized as histograms or fed into automation.
- For instance, it would allow:
  - showing a histogram of message size for sent/received network communication, or
  - showing a sum of error counts per error type, or
  - analyzing the efficiency of layers of software cache mechanisms, counting the number of cache hit vs misses.

Fast trace control feedback loop

- Objectives: [**relevance**]
- Fast tracing start and stop
  - Introduce a fast start/stop flag in shared memory
  - Quickly react to specific events to immediately trace with more or less details
- Fast counter-based trace control
  - Track resources across many CPUs
  - Hierarchical carry propagation tree provides a fast approximation
  - Trace hit counters extension to provide immediate trace filtering feedback loop based on counter sum approximation

Reduce LTTng Memory Usage

- Objectives: [**overhead, configuration**]
- Limiting memory to N tracing buffers without limiting which N CPUs are used
- Concurrency IDs for containers
- Maximum concurrency ID limits per container
- Relevant use-case: containers restricted by CPU time and number of threads rather than by cpusets

Improve LTTng static instrumentation API

- Objectives: [**instrumentation, trace model**]
- New libside instrumentation API and SIDE ABI specification
- Benefits:
  - Improved error reporting compared to LTTng-UST tracepoints
  - Application state dumps
  - Instrumentation of other languages/runtimes
  - Integration with other tracers
  - Compiler-based static type checker
  - Simpler and more efficient RCU implementation than the implementation in LTTng-UST

Dynamic Instrumentation

- Objectives: [**overhead, instrumentation**]
- Add fast dynamic instrumentation
- *Libpatch* enables dynamic instrumentation of userspace applications with low overhead.
- It achieves results similar to Dyninst with a fraction of the runtime latency when inserting the instrumentation on a live process.
- Integrating it with LTTng-UST and libside would allow end users to augment the information gathered by static instrumentation with dynamic instrumentation.
- Integration with DWARF would allow specifying which variables should be captured as payload.

Reduce Userspace Code Patching Overhead

- Objectives: [**overhead, instrumentation**]
- Reduce overhead of code patching for static and dynamic instrumentation.
- Proposing a new “pokev” Linux system call
  - Takes care of userspace code patching without losing executable page sharing across processes due to Copy-on-Write (CoW).
  - Eliminates significant overhead in terms of memory and CPU cache use when instrumenting core libraries which are used by many processes by preventing each process from allocating its own copy of the modified pages.
  - Handle XMC (cross-CPU code modification) architectural requirements.

Summary

Tracing Challenges

Minimizing resource usage...

- Memory footprint
- CPU overhead

...while extracting helpful data.

Recent LTTng R&D

Minimizing resource usage...

- Memory footprint (LTTng 2.14, 2.15, 2.16)
- CPU overhead (LTTng 2.16)

...while extracting helpful data. (LTTng 2.16)

LTTng Releases

LTTng 2.14

- Per-channel buffers

LTTng 2.15 – Q1 2026

- Further improve memory footprint
- Improve tracing buffer robustness
- Produce CTF 2 traces

LTTng 2.16 – Q2 2026

- Add Aggregation Maps (with Trace Hit Counters)

Interested in more about...

Efficient memory usage of tracing buffers?

Improved impact of tracing tools?

Other topics?

Come speak with us at tomorrow's hackathon!

Questions ?

- Links:
  - https://www.efficios.com
  - https://lttng.org
  - https://babeltrace.org
  - https://diamon.org
  - https://barectf.org

Contacts

Mathieu Desnoyers – mathieu.desnoyers@efficios.com

Jérémie Galarneau – jgalar@efficios.com

Erica Bugden – ebugden@efficios.com

Annex

References

- Common Trace Format 2 Specification

  https://diamon.org/ctf

- libside repository

  https://github.com/efficios/libside

Field classes common to CTF 1 and CTF 2

Field class	CTF 1.8	CTF 2
Fixed-length integer	✓	✓
UTF-8 string	✓	✓
Floating point number	✓	✓
Fixed-length array	✓	✓
Dynamic-length array	✓	✓
Structure	✓	✓
Variant	✓	✓

What does CTF 2 do better than CTF 1?

	CTF 1.8	CTF 2
Metadata format	TSDL (custom DSL) • Non-trivial to parse.	JSON text sequences • Widely used standard format with pre-existing parser libraries in various languages.
Augment events and fields with user-defined metadata	✗	✓ • Associate user-defined name to a value. • Used to tailor analysis or pretty printing of trace data.

What does CTF 2 do better than CTF 1?

Field class	CTF 1.8	CTF 2
BLOB	✗	✓ • Record opaque binary blobs • IANA media type attribute
Optional	✗	✓
LEB128 variable length integer	✗	✓ • Values > 64-bit range • Common need in scientific computing
UTF-16 and UTF-32 string character encoding	✗	✓ • Native string encoding on some platforms • E.g. Windows, Java VM
Fixed-length bit map	✗	✓ • Associate names to specific bits in a bitmap • Useful to represent flags
Boolean	✗	✓

Linux Kernel & Community Work

Laying the foundation to...

Reduce userspace tracer CPU and memory overhead

- Reduce CPU execution constraints by replacing hardware atomic instructions with kernel-managed software transactions
  - Restartable sequences (RSEQ) system call and GNU C library integration
- Bound memory allocation to max number of concurrently running threads (rather than allocate for each CPU)
  - RSEQ concurrency IDs (mm_cid)
- Reduce CPU data cache & branch prediction buffer impact of static instrumentation
  - Concurrent code patching (XMC), page deduplication
  - Also useful for code specialization at runtime

Linux Kernel & Community Work

Laying the foundation to...

Enable kernel tracer to have previously unavailable data

- Handle page faults while tracing system calls
  - Faultable system call tracepoints

Expand integration of LTTng-UST with the open source ecosystem

- Instrumentation coverage of runtimes, libraries, applications
- Tracer-agnostic "SIDE" instrumentation specification
- libside reference implementation for C/C++
- ABI targets instrumentation of various runtimes natively

SIDE ABI RFC (libside)

- The SIDE ABI is currently at RFC stage, aiming to create a specification.
  - https://github.com/efficios/libside/blob/master/doc/rfc-side-abi.txt
- Runtime/language agnostic,
- Supports multiple concurrent tracers,
- Instrumentation is not specific to a tracer,
  - No need to rebuild applications if using a different tracer,
- Instrumentation can be either static or dynamic,
- Supports complex/nested types,
- Supports both static and dynamic types,
- libside is a C/C++ reference implementation for the System V ELF ABI.