# Exa-Tracer

## Tracing HPC Supercomputers with LTTng

EfficiOS

# What is Tracing?

- **Instrumentation** of code:
  - Static insertion & dynamically enabled,
  - Fully dynamic.
- Collect or react to a **sequence of events** emitted during code execution:
  - Minimal intrusiveness on the workload,
  - **Low-overhead** is key.
- Can be compared to logging, except that it needs to handle a **high event throughput** (millions events per second).

# Why Tracing?

- **Identify root cause** of:
  - Heisenbugs
    - Hard to reproduce issues,
    - Issues that disappear under observation.
  - Performance bottlenecks
  - Protocol, API, ABI specification violation
- **Monitor** a program behavior and **react** when detecting:
  - performance outliers,
  - errors (e.g. application core dump).

# Tracing and Profiling

- *Profiling* and *Tracing* are **complementary** diagnostic techniques
- *Profiling is* good at identifying **active usage** of resources
- *Tracing* excels at identifying **resources misuse** (*e.g.* wait time, threads blocked on synchronization or I/O)

# LTTng Historic Focus

- **LTTng** (2005-) [1]:
  - Telecom
  - **Embedded/real-time** systems
  - Large **multi-core** systems
  - **Linux kernel** and applications
  - Common Trace Format [2] (**CTF**)
  - Developed in collaboration with trace analysers
    - **Babeltrace** [3]
    - **Trace Compass** [4]

# Common Trace Format: 1.8 (2012)

- Domain specific language metadata
  - **Structured** type system
  - Can be defined **statically** or **dynamically**
  - Clock descriptions for **correlation** between traces
- Binary trace format
  - **Compact**
  - Fast to **produce**
  - Easily generated by **software** and **hardware** components

EfficiOS

# Common Trace Format: 2 (2024)

- **JSON** metadata
  - Superset of CTF 1.8 type system
  - New built-in types
  - User defined metadata and extensions
- Binary trace format
  - Superset of CTF 1.8 trace format
  - Support the new types

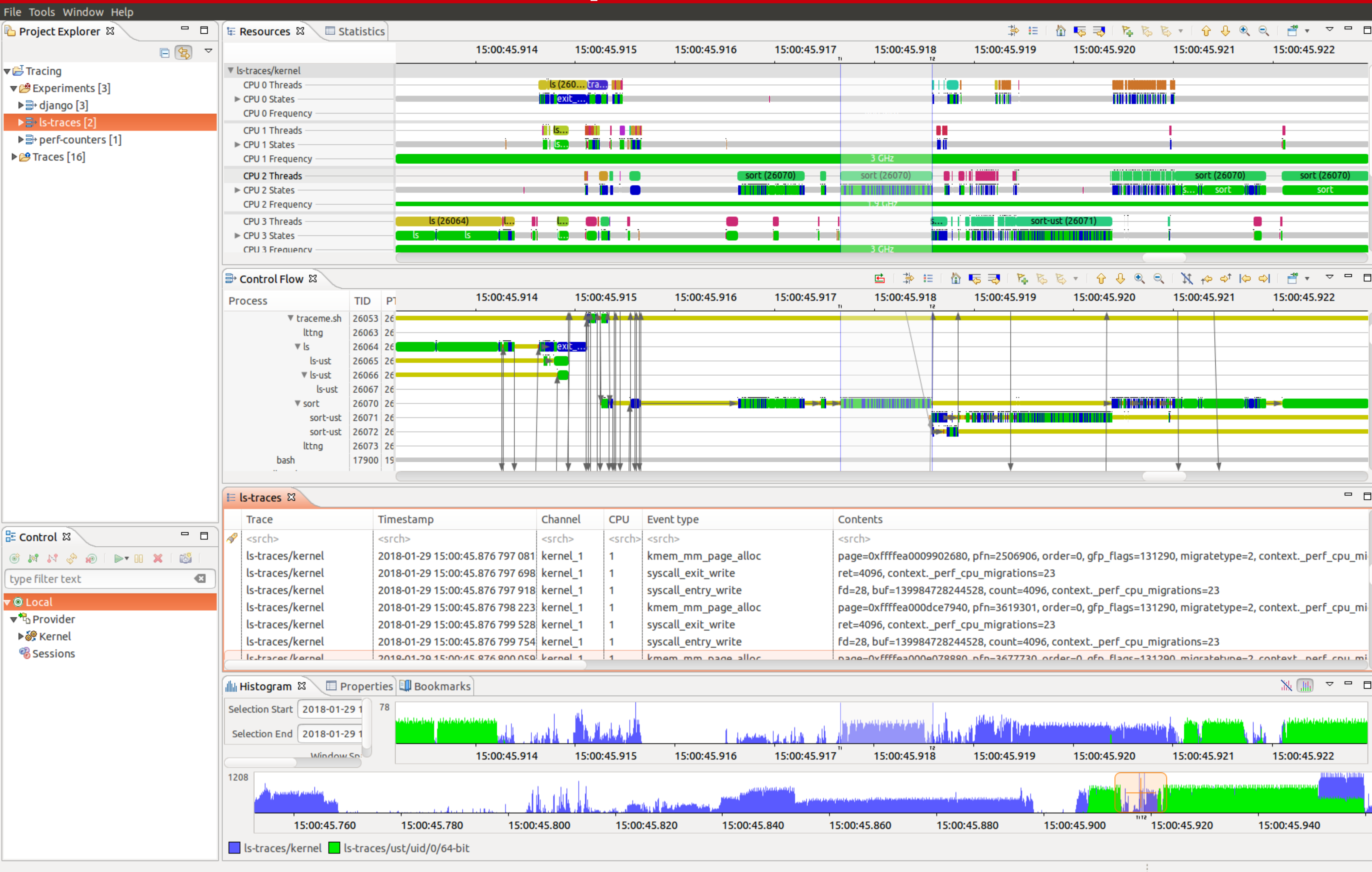
# Babeltrace

- **Trace reader**:
  - Convert, filter, seek and **analyse** traces
- **Plugin system**
  - Allow supporting arbitrary trace formats
  - Allow adding custom phases (*e.g.* new analysis, filtering)
  - Built-in plugins for **CTF 1.8 and 2**
- C/C++ library API
- Python bindings
- Command line interface

# Trace Compass

- Support **CTF** and other trace formats
- Plugin system for views and analysis phases
- Many built-in views and analysis
  - Linux kernel
    - Disk I/O
    - Scheduler
    - Hardware resources (e.g. CPU, IRQ)
    - And many more
  - **Critical path analysis**
  - **Log** correlation with traces
  - **Network packet** correlation

# Trace Compass Kernel View



# LTTng Main Features

- Low-overhead
  - Fast per-CPU ring buffer
    - ~100 ns/event
  - User-configurable dynamic runtime filters
    - ~50 ns/evaluation
- Event streaming (disk or network)
- Snapshot mode
  - Flight recorder in memory
- Triggers
- Event notifications with payload capture
- Session rotation

# LTTng Trace Correlation

- Correlation across tracing domains:
  - Linux kernel
  - Userspace
- Correlation across hosts
  - based on NTP/PTP time synchronization
    OR
  - realign traces at post-processing based on network communications

# HPC Collaboration: ANL

- Argonne National Laboratory
  - **THAPI** (developed by ANL) [5,6]
  - Based on **LTTng** and **Babeltrace**
  - Instrumentation of OpenCL, Level Zero, Cuda Runtime, HIP, OMPT.
  - Developed for tracing Aurora
    - 10 624 nodes
    - 9 264 128 cores
    - Capture millions of events/s per node
    - Run for hours with tracing enabled

# HPC Collaboration: LLNL

- Lawrence Livermore National Laboratory
  - **Exa-Tracer** [7] (developed by EfficiOS in collaboration with AMD)
  - Instrumentation of:
    - ROCm (HIP, HSA, ROC-TX, GPU kernels dispatch)
    - OpenMPI, CrayMPI.
  - Developed for tracing El Capitan:
    - 11 000 nodes
    - 11 039 616 cores

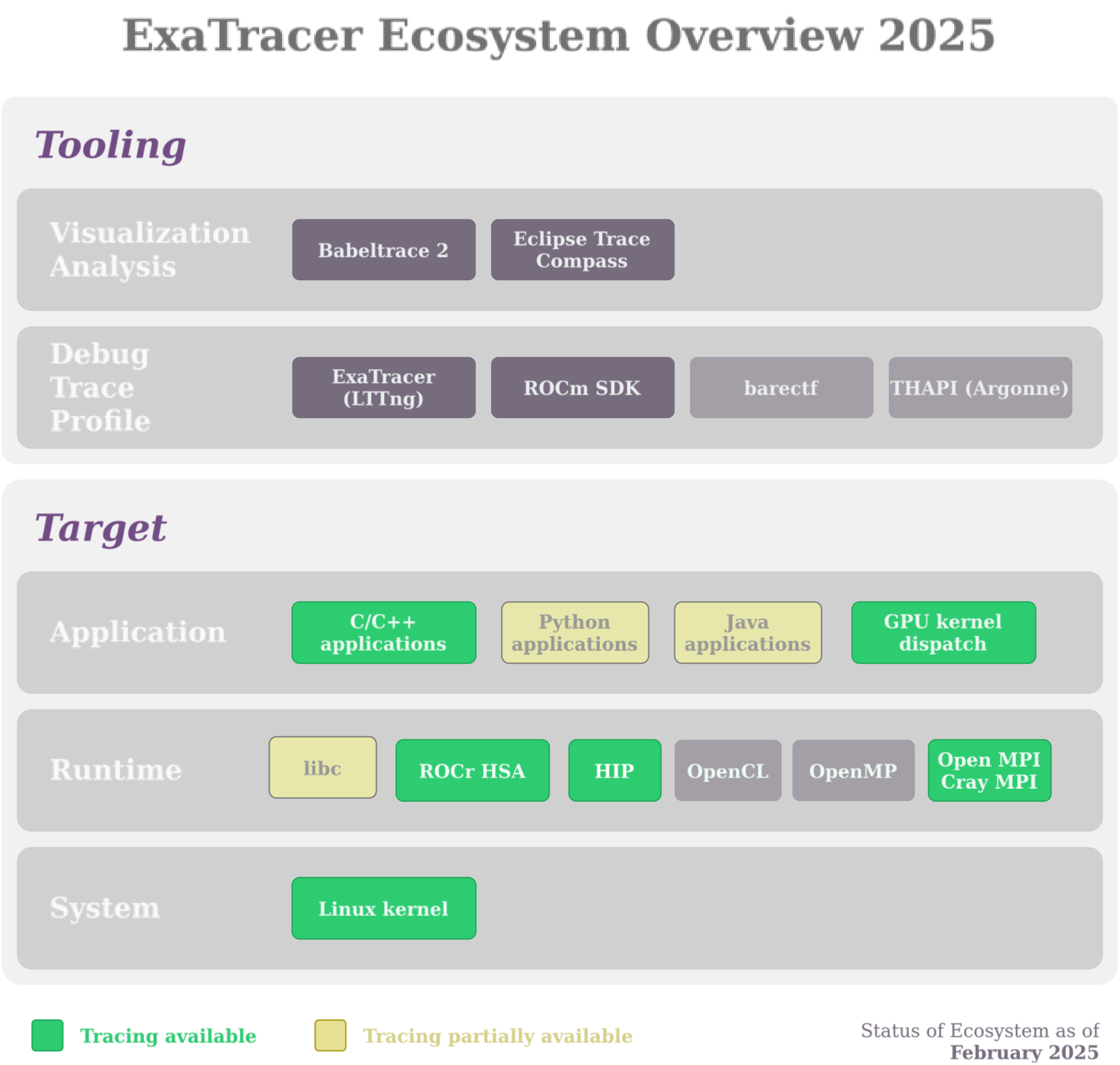
EfficiOS

# HPC Collaboration: Polytechnique Montréal

- École Polytechnique de Montréal
  - **Trace Compass**
  - Trace analyser and visualizer
  - Working on improvement of scalability to large traces (100 GB+)
    - Reduce reaction time (interactivity)
    - Reduce analysis delay

EfficiOS

# HPC Software Stack Diagram



# Exa-Tracer

- Targeting MPI, HSA, HIP, ROC-TX and GPU kernels dispatches
- Header files parsed with Clang
  - For MPI, HSA and HIP
  - Instrumentation wrappers generated automatically
- HIP/HSA instrumented via interception tables
- MPI and ROC-TX instrumented with *LD_PRELOAD* symbols override
- GPU kernels instrumented using rocprofiler-sdk

# Babeltrace Text Output (HPC)

```
[14:32:13.124735003] (+?.?????????) epycamd mpi:enter_MPI_Init: { cpu_id = 16 }, { _app_MPI_rank_tag = ( "_none" : container = 0 ), _app_MPI_rank = { { } } }, { lttng_thr
[14:32:13.134930023] (+0.010195020) epycamd mpi:enter_MPI_Init: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_none" : container = 0 ), _app_MPI_rank = { { } } }, { lttng_thre
[14:32:13.377854633] (+0.242924610) epycamd mpi:exit_MPI_Init: { cpu_id = 0 }, { _app_MPI_rank_tag = ( "_none" : container = 0 ), _app_MPI_rank = { { } } }, { lttng_threa
[14:32:13.377935039] (+0.000080406) epycamd mpi:enter_MPI_Comm_rank: { cpu_id = 0 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_
[14:32:13.377936789] (+0.000001750) epycamd mpi:exit_MPI_Comm_rank: { cpu_id = 0 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_t
[14:32:13.377940489] (+0.000003700) epycamd mpi:enter_MPI_Comm_size: { cpu_id = 0 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_
[14:32:13.377941559] (+0.000001070) epycamd mpi:exit_MPI_Comm_size: { cpu_id = 0 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_t
[14:32:13.377948109] (+0.000006550) epycamd mpi:exit_MPI_Init: { cpu_id = 17 }, { _app_MPI_rank_tag = ( "_none" : container = 0 ), _app_MPI_rank = { { } } }, { lttng_thre
[14:32:13.378089333] (+0.000141224) epycamd mpi:enter_MPI_Comm_rank: { cpu_id = 17 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng
[14:32:13.378092072] (+0.000002739) epycamd mpi:exit_MPI_Comm_rank: { cpu_id = 17 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_
[14:32:13.378096722] (+0.000004650) epycamd mpi:enter_MPI_Comm_size: { cpu_id = 17 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng
[14:32:13.378110662] (+0.000013940) epycamd mpi:exit_MPI_Comm_size: { cpu_id = 17 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_
[14:32:13.385016917] (+0.006906255) epycamd hip:hipMalloc_entry: { cpu_id = 17 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_thr
[14:32:13.388492683] (+0.003475766) epycamd hip:hipMalloc_entry: { cpu_id = 0 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_thre
[14:32:13.435549095] (+0.047056412) epycamd hip:hipMalloc_exit: { cpu_id = 16 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_thre
[14:32:13.435559024] (+0.000009929) epycamd hip:hipMemcpy_entry: { cpu_id = 16 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_thr
[14:32:13.458713481] (+0.023154457) epycamd hip:hipMalloc_exit: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_threa
[14:32:13.458724041] (+0.000010560) epycamd hip:hipMemcpy_entry: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_thre
[14:32:13.775863363] (+0.317139322) epycamd hip:hipMemcpy_exit: { cpu_id = 2 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_threa
[14:32:13.775881682] (+0.000018319) epycamd mpi:enter_MPI_Send: { cpu_id = 2 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_threa
[14:32:13.783083534] (+0.007201852) epycamd hip:hipMemcpy_exit: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_threa
[14:32:13.783095603] (+0.000012069) epycamd mpi:enter_MPI_Recv: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_threa
[14:32:13.783155251] (+0.000059648) epycamd mpi:exit_MPI_Recv: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_thread
[14:32:13.783156521] (+0.000001270) epycamd mpi:exit_MPI_Send: { cpu_id = 2 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_thread
[14:32:13.783157291] (+0.000000770) epycamd hip:hipMemcpy_entry: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_thre
[14:32:13.783168190] (+0.000010899) epycamd hip:hipFree_entry: { cpu_id = 2 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_thread
[14:32:13.783188849] (+0.000020659) epycamd hip:hipFree_exit: { cpu_id = 2 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_thread_
[14:32:13.783193839] (+0.000004990) epycamd mpi:enter_MPI_Finalize: { cpu_id = 2 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_t
[14:32:13.783768544] (+0.000574705) epycamd hip:hipMemcpy_exit: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_threa
[14:32:13.783779623] (+0.000011079) epycamd hip:hipFree_entry: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_thread
[14:32:13.783804182] (+0.000024559) epycamd hip:hipFree_exit: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_thread_
[14:32:13.783808692] (+0.000004510) epycamd mpi:enter_MPI_Finalize: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_t
[14:32:13.828399162] (+0.044590470) epycamd mpi:exit_MPI_Finalize: { cpu_id = 1 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 1 } }, { lttng_th
[14:32:13.828668650] (+0.000269488) epycamd mpi:exit_MPI_Finalize: { cpu_id = 2 }, { _app_MPI_rank_tag = ( "_int64" : container = 4 ), _app_MPI_rank = { 0 } }, { lttng_th
```

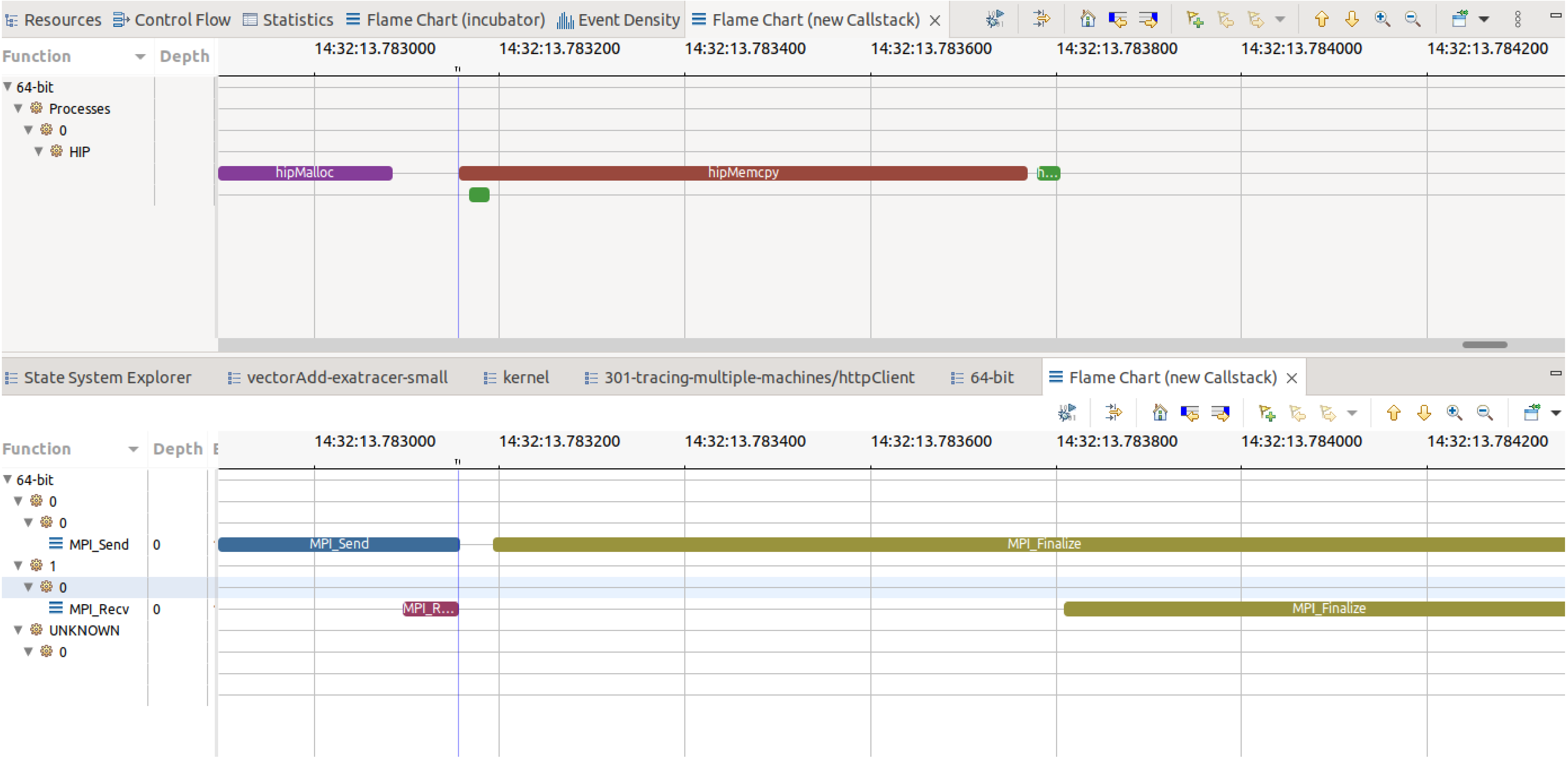
# Babeltrace Text Output (HPC)

```
[14:32:13.385016917]              ; Timestamp
(+0.006906255)                    ; Relative in time (ns) to the previous event
epycamd                           ; Hostname
hip:hipMalloc_entry:              ; provider:tracepoint
; Context fields:
{ cpu_id = 17 },                  ;   CPU on which the event was emitted
{ ...  _app_MPI_rank = { 1 } }, ;   MPI rank on which the event was emitted
; Event payload:
{ lttng_thread_id = 0,            ;   Unique thread ID (not OS)
lttng_local_id = 0,               ;   Unique per-thread ID for entry/exit correlation
ptr = 0x7FFD84572790,             ;   Pointer filled with hipMalloc() allocation
size = 400 }                      ;   Asked size of allocation hipMalloc()

[14:32:13.783156521]              ; Timestamp
(+0.000001270)                    ; Relative in time (ns) to the previous event
epycamd                           ; Hostname
mpi:exit_MPI_Send:                ; provider:tracepoint
; Context fields:
{ cpu_id = 2 },                   ;   CPU on which the event was emitted
{ ... _app_MPI_rank = { 0 } },  ;   MPI rank on which the event was emitted
; Event payload:
{ lttng_thread_id = 0,            ;   Unique thread ID (not OS)
lttng_local_id = 3,               ;   Unique per-thread ID for entry/exit correlation
lttng_has_ret = 1,                ;   Did the function returns (no exception)?
lttng_ret = 0 }                   ;   Returned value (valid only if lttng_has_ret = 1)
```

EfficiOS

# Trace Compass (HPC)




EfficiOS

# Challenges of Tracing HPC Clusters

- Large volume of data
  - Execution overhead
  - Memory footprint and bandwidth
  - I/O throughput
  - Storage
- Waiting time between trace generation and visualisation of analysis results
- Interactivity of trace visualisation at scale (100GB+ traces)
- Precision of trace correlation across hosts

# Future Work (Instrumentation)

- Improve instrumentation coverage granularity:
  - API annotations
    - Function arguments input/output/in-out
    - Tagged unions
- SIDE [8]
  - ABI defining an extensible type system for instrumentation
  - Support nested compound types
  - Runtime and OS agnostic

EfficiOS

# Future Work (Trace Analysis)

- Improve interactivity of Trace Compass for large traces
  - State History Scalability
- Reduce delay between production and availability of analysis results
  - Partition trace analysis
    - Node-local vs Global interactions
  - Pipeline trace analysis

# References


1. https://lttng.org/
2. http://diamon.org/ctf/
3. https://babeltrace.org/
4. https://eclipse.dev/tracecompass/
5. https://github.com/argonne-lcf/THAPI
6. https://tracingsummit.org/ts/2023/files/Heterogeneous_Appencourt_Videau.pdf
7. https://git.efficios.com/deliverable/exatracer.git
8. https://github.com/efficios/libside


EfficiOS