Updates on Nodejs Performance Analysis

Progress Report Meeting

Hervé KABAMBA

PhD Candidate

Supervisor: Michel Dagenais

December 08, 2022

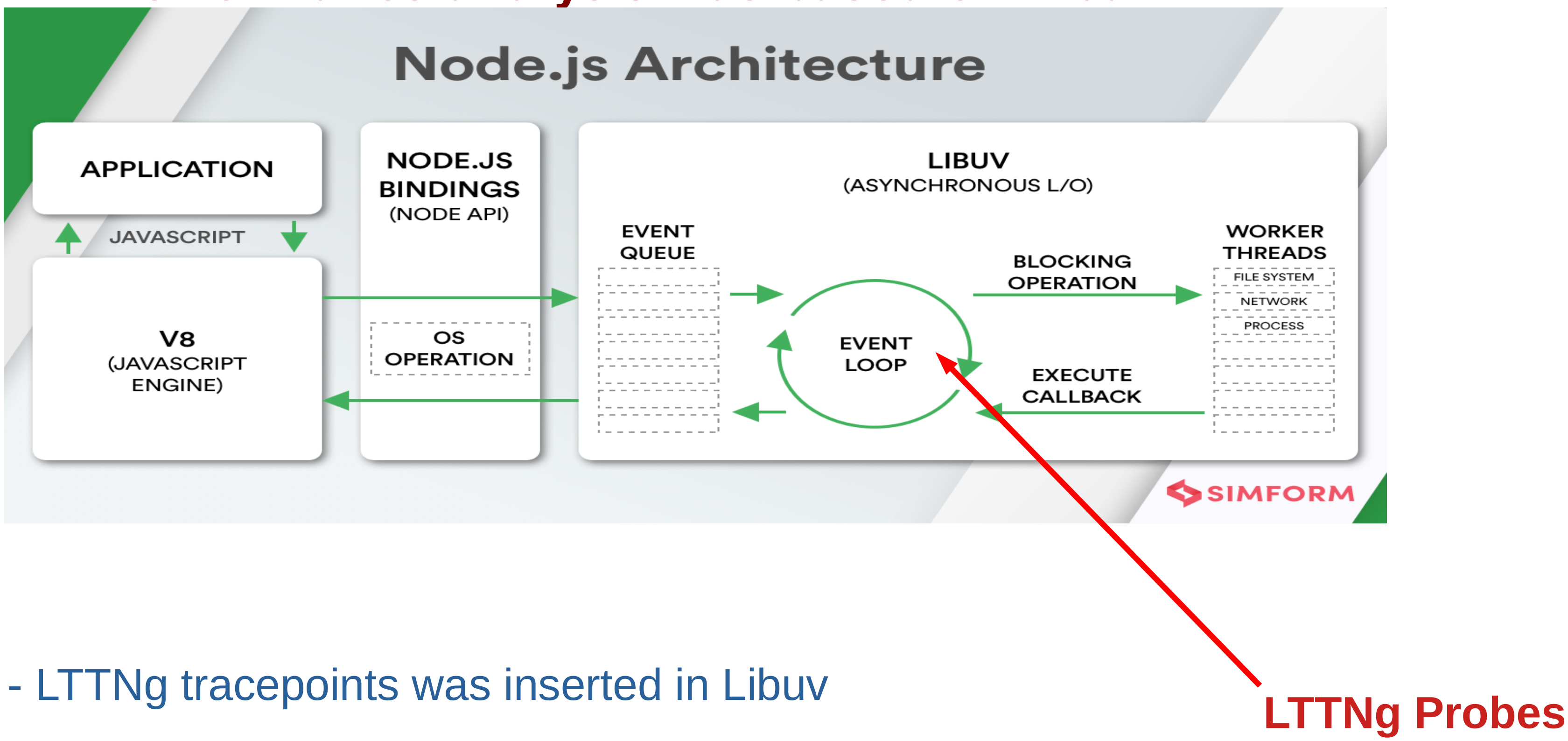
Polytechnique Montréal

Département de Génie Informatique et Génie Logiciel

Agenda

Previous work

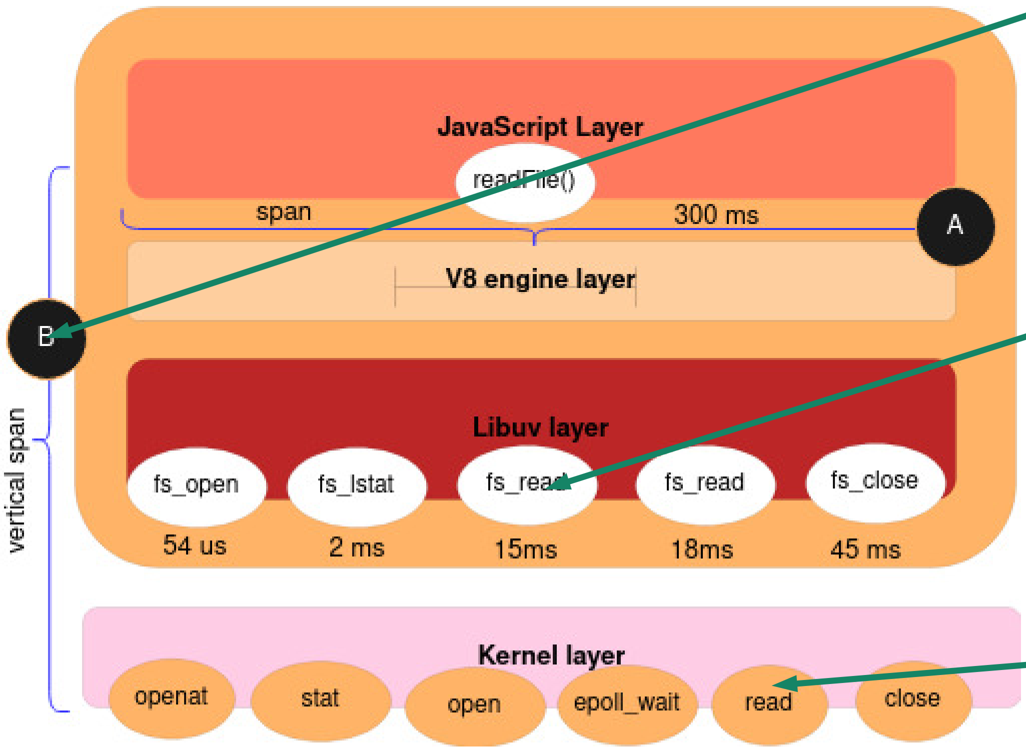
Performance analysis was based on Libuv



- LTTNg tracepoints was inserted in Libuv
- Event-loop statuses could be tracked
- Asynchronous operations life-cycle could be tracked
- Lttng tracepoints could be inserted into Javascript

Previous Work

Vertical Span Representation Model



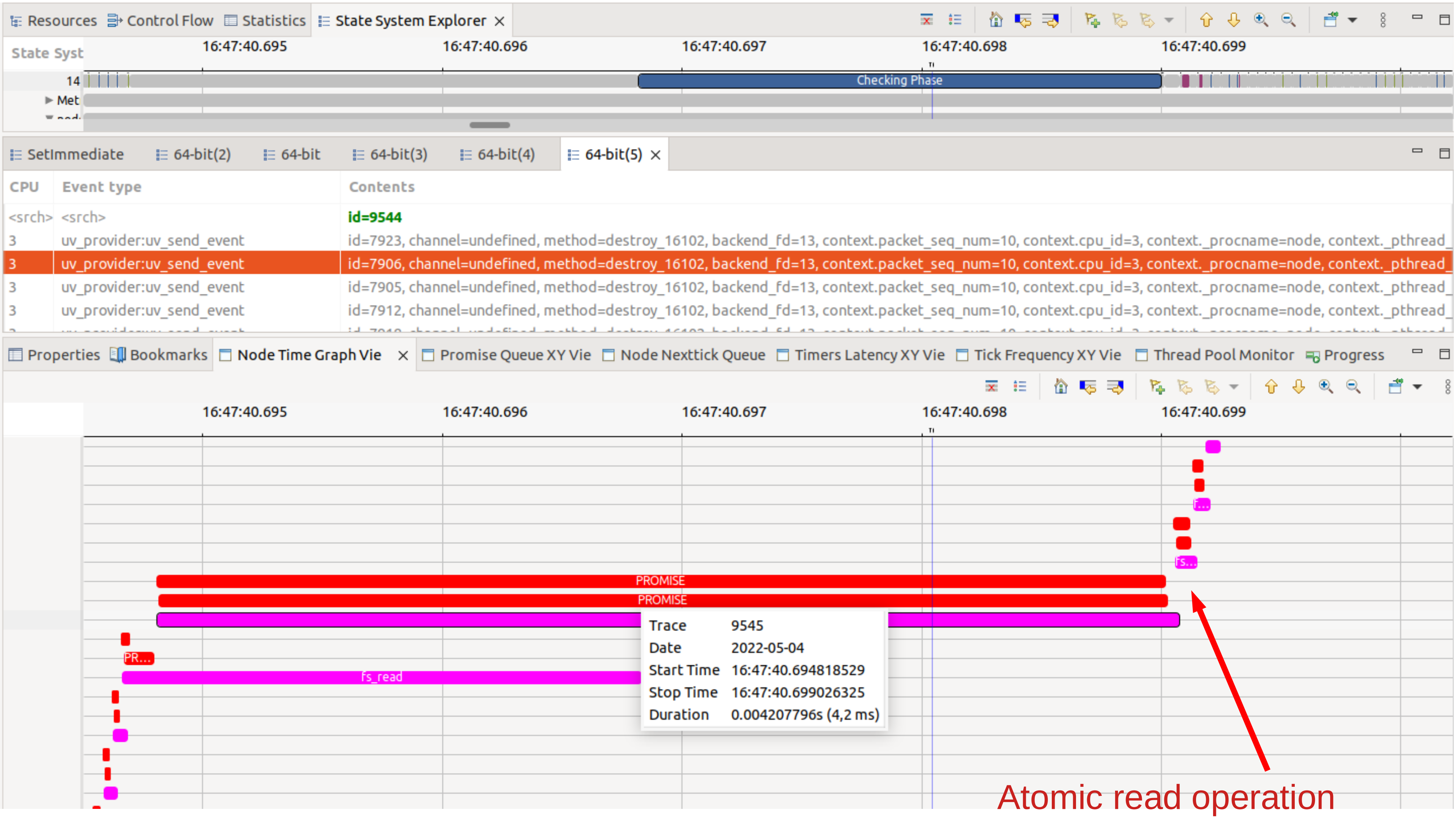
Introduce Vertical Span Model (VSM)

Consider atomic operations in Node

Map atomic operations with Syscalls

Previous Work

Example of previous Analysis



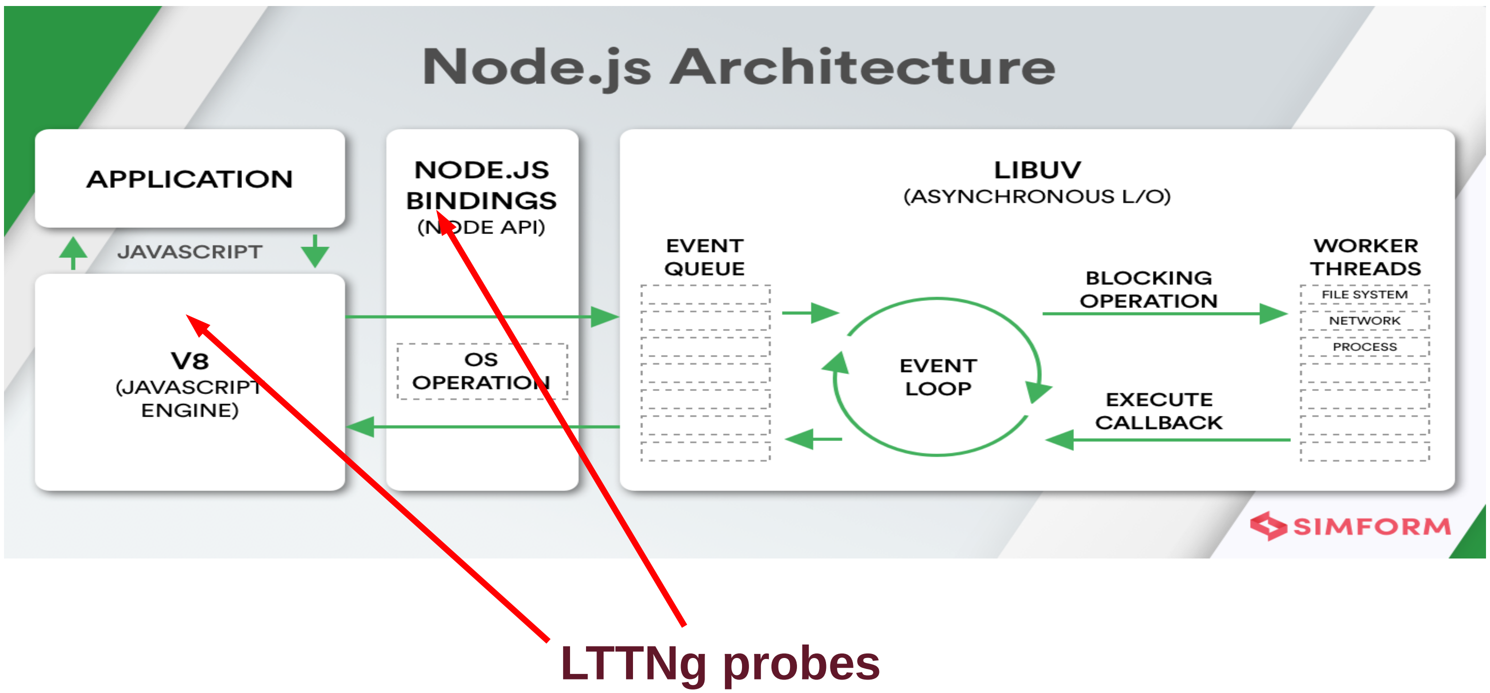
Data Collection improvements

Three levels:

Nodejs Compilation process

Nodejs VMs   (V8 engines instances)

Nodejs C++ Bindings

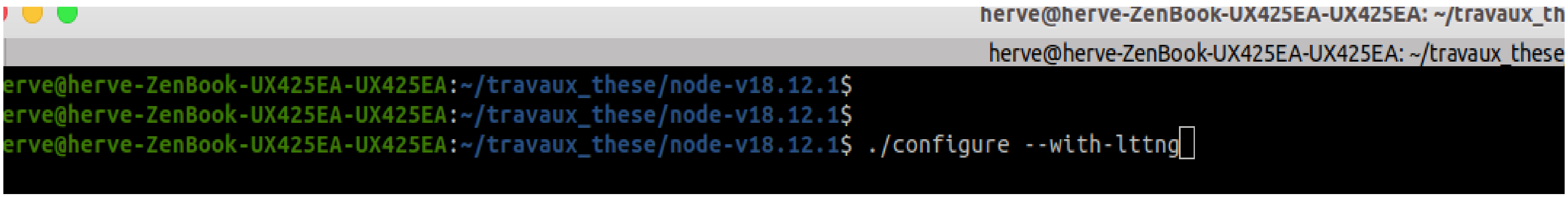


Data Collection improvements

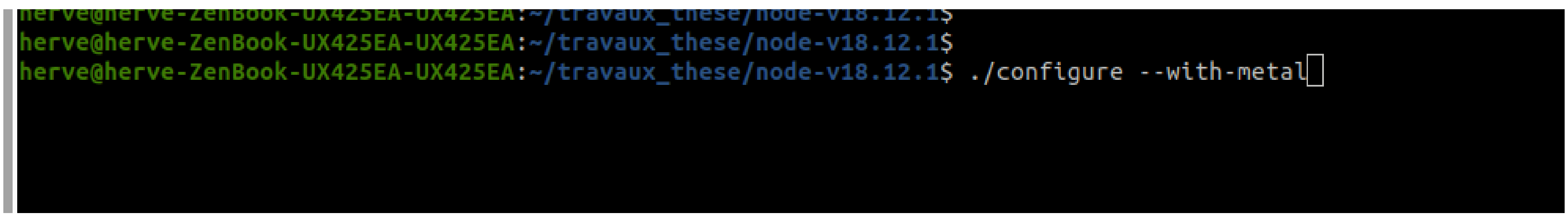
At compilation level:

Macros are defined to support multi-platforms compilation process

On Linux, configure the source with ***–with-lttng*** flag to activate the probes



On other platforms such as Windows, configure the source with ***–with-metal*** flag to activate the probes

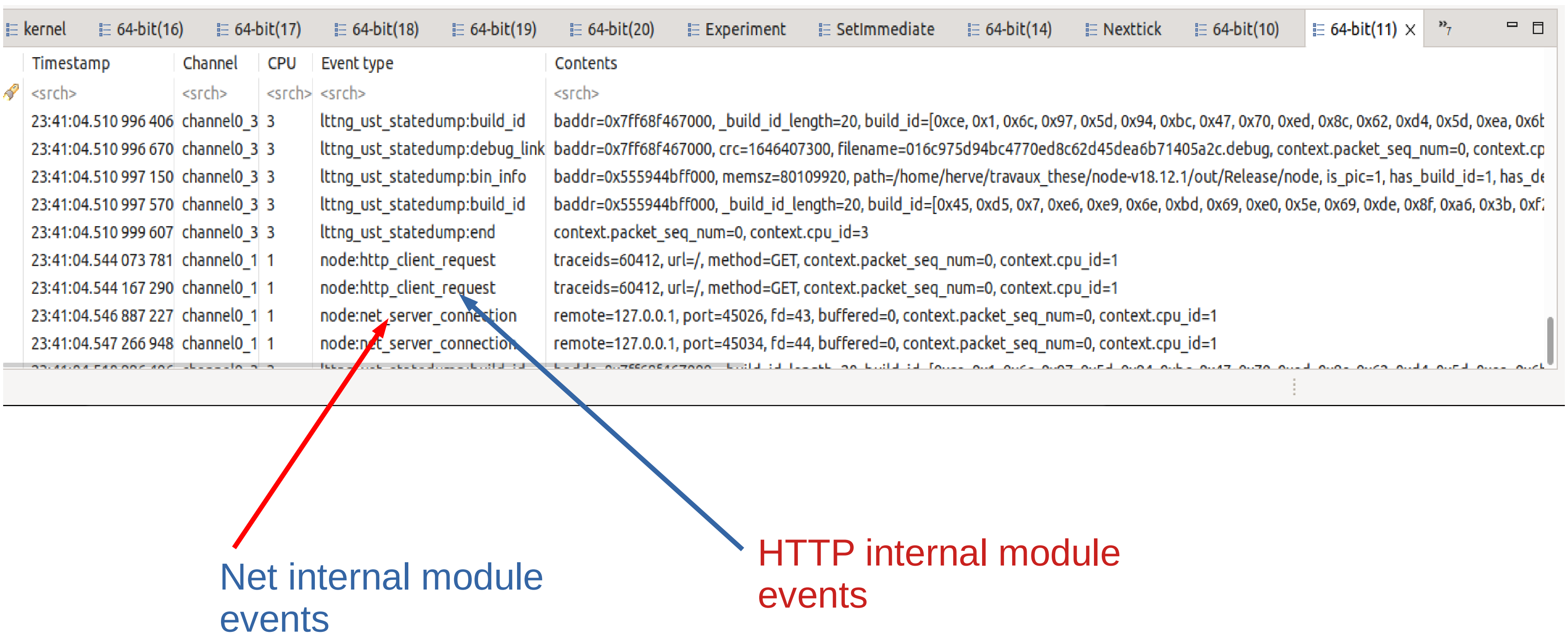


Data Collection improvements

- No need for high level instrumentation (Javascript)

- Everything is handled by the VM and bindings probes

- Everything is transparent from the user



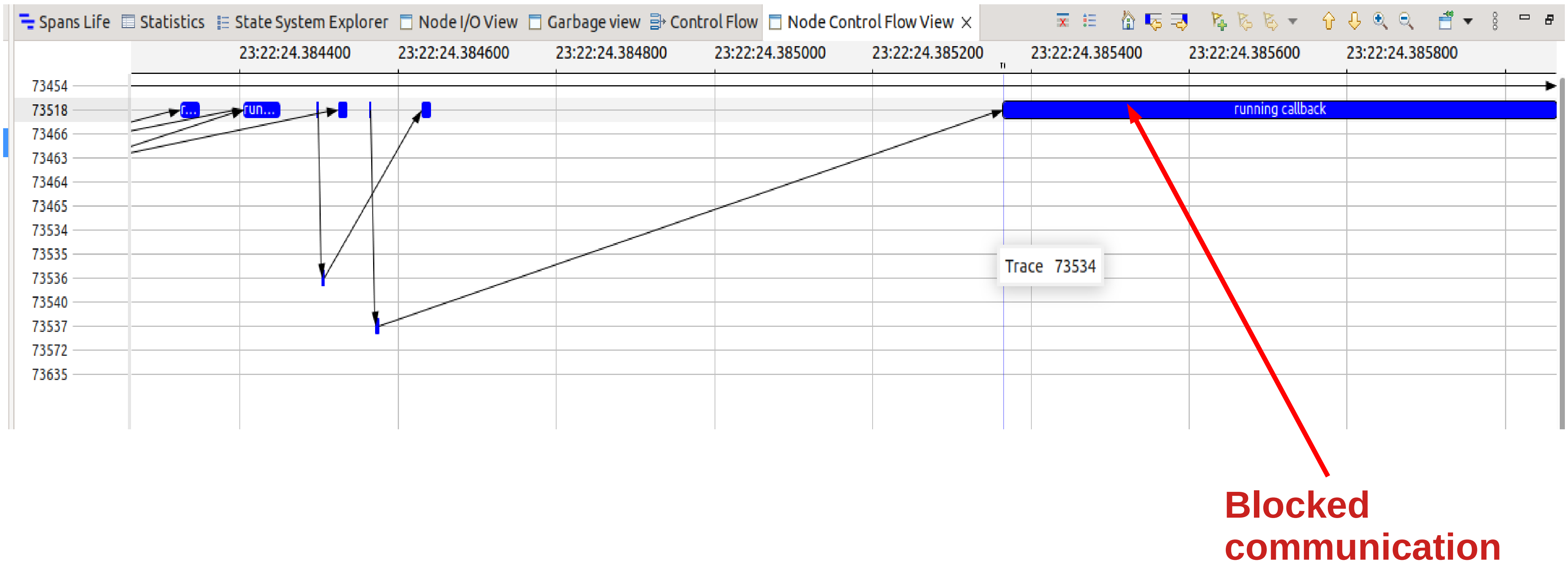
Use Cases

1. Windows Nodejs Inter-process communication

- Fork a new process on windows from Nodejs
- **const child = fork('./fork.js', [], { stdio: ['inherit', 'inherit', 'inherit', 'ipc', 'pipe'] });**
- Communicate with the forked process by sending ping
- The main process expects to receive pong replies form the fork
- However, no answer comes out.

Use Cases

We track the inter-process communication on windows



- Windows I/O operations can be synchronous or asynchronous

- No abstraction of the platform by Libuv for the pipe socket

- You have to indicate that you deal with Windows by adding the **“overlapped”** flag in the fork parameters, to enforce windows piping

Use Cases

2. Memory leaks tracking in VMs

-We track the garbage collection triggers

- Define some metrics:
  * Time spent in the GC (**TIGC**), Time between 2 GC operations (**TBGC**)

- If **TIGC > TBGC:** Application is seriously starving**(Probable memory leak)**

Use Cases

Steps to reproduce the use case 2

- Run a faulty Nodejs application

- The app. starts to consume much memory until the an OOM error is triggered

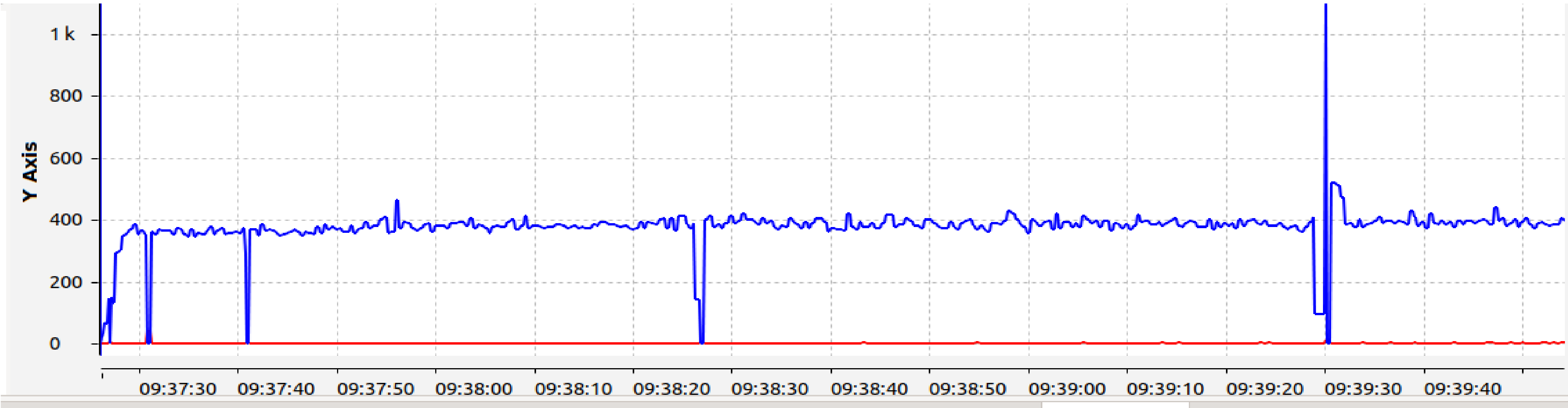
- Observe the patterns in Trace Compass

```
herve@herve-ZenBook-UX425EA-UX425EA: ~/travaux_these/node-v18.12.1 211x25
[328398:0x5568fb25ad20]    21663 ms: Mark-sweep (reduce) 198.3 (202.5) -> 198.0 (202.5) MB, 196.9 / 0.0 ms  (+ 0.1 ms in 2 steps since start of marking, biggest step 0.0 ms, walltime since start of marking 202 m
s) (average mu = 0.212, current mu = 0.179) finalize incremental marking via stack guard; GC in old space requested
~~> 8656232 entries to recordrecord
<--- Last few GCs --->

[328398:0x5568fb25ad20]    21663 ms: Mark-sweep (reduce) 198.3 (202.5) -> 198.0 (202.5) MB, 196.9 / 0.0 ms  (+ 0.1 ms in 2 steps since start of marking, biggest step 0.0 ms, walltime since start of marking 202 m
s) (average mu = 0.212, current mu = 0.179)

<--- JS stacktrace --->

FATAL ERROR: Ineffective mark-compacts near heap limit Allocation failed - JavaScript heap out of memory
 1: 0x5568f4dba6f4 node::Abort() [./node]
 2: 0x5568f4ca5725  [./node]
 3: 0x5568f4fc6e74 v8::Utils::ReportOOMFailure(v8::internal::Isolate*, char const*, bool) [./node]
 4: 0x5568f4fc718f v8::internal::V8::FatalProcessOutOfMemory(v8::internal::Isolate*, char const*, bool) [./node]
 5: 0x5568f51d4919  [./node]
 6: 0x5568f51d58f6 v8::internal::Heap::RecomputeLimits(v8::internal::GarbageCollector) [./node]
 7: 0x5568f51e76bc v8::internal::Heap::PerformGarbageCollection(v8::internal::GarbageCollector, v8::internal::GarbageCollectionReason, char const*, v8::GCCallbackFlags) [./node]
 8: 0x5568f51e8335 v8::internal::Heap::CollectGarbage(v8::internal::AllocationSpace, v8::internal::GarbageCollectionReason, v8::GCCallbackFlags) [./node]
 9: 0x5568f51eb797 v8::internal::Heap::HandleGCRequest() [./node]
10: 0x5568f5162334 v8::internal::StackGuard::HandleInterrupts() [./node]
11: 0x5568f55ea88a v8::internal::Runtime_StackGuard(int, unsigned long*, v8::internal::Isolate*) [./node]
12: 0x5568f5a92439  [./node]
Abandon (core dumped)
herve@herve-ZenBook-UX425EA-UX425EA:~/travaux_these/node-v18.12.1$
```

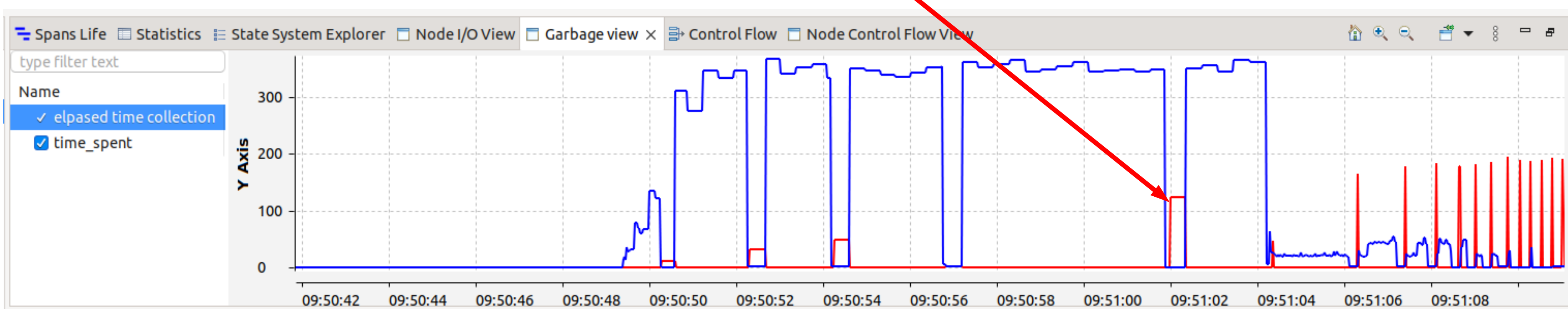
Use Cases

Memory leaks tracking in VMs

Healthy application patterns



Application starving due to memory leaks **TIGC > TBGC**



Conclusion

- Performance analysis process improvements with user-less JavaScript instrumentation

- VM performance analysis

- Inter-process communication tracking

- Compilation improvements for muti-platform support

Ongoing :

- Isolate faulty user-level code (memory leaks sources)

-Expose more performance counters related to VM Perf. Analysis

Bibliography

[1] I. Beschastnikh, P. Wang, Y. Brun, M. D. Ernst, Debugging distributed systems, ACM-Queue (2015).

[2] J. Hoglund, An analysis of a distributed tracing systems effect on performance. jaeger and opentracing api, UMEA University (2020).

[3] S. Tilkov, S. Vinoski, Node.js: Using javascript to build high-performance network programs, IEEE INTERNET COMPUTING (2010).

[4] Cloud desktop ide platform.

URL https://kubernetes.io/fr/

[5] Visual studio code.

URL https://code.visualstudio.com/

[6] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, A. Vahdat, Exploiting a natural network effect for scalable, fine-grained clock synchronization, 2018.

2007, pp. 171–180.

Thank you