SELA

SELA | DEVELOPER | PRACTICE

July 3-5, 2018

Kevin Gosse @kookiz
Grégory Léocadie @gleocadie
Christophe Nasarre @chnasarre

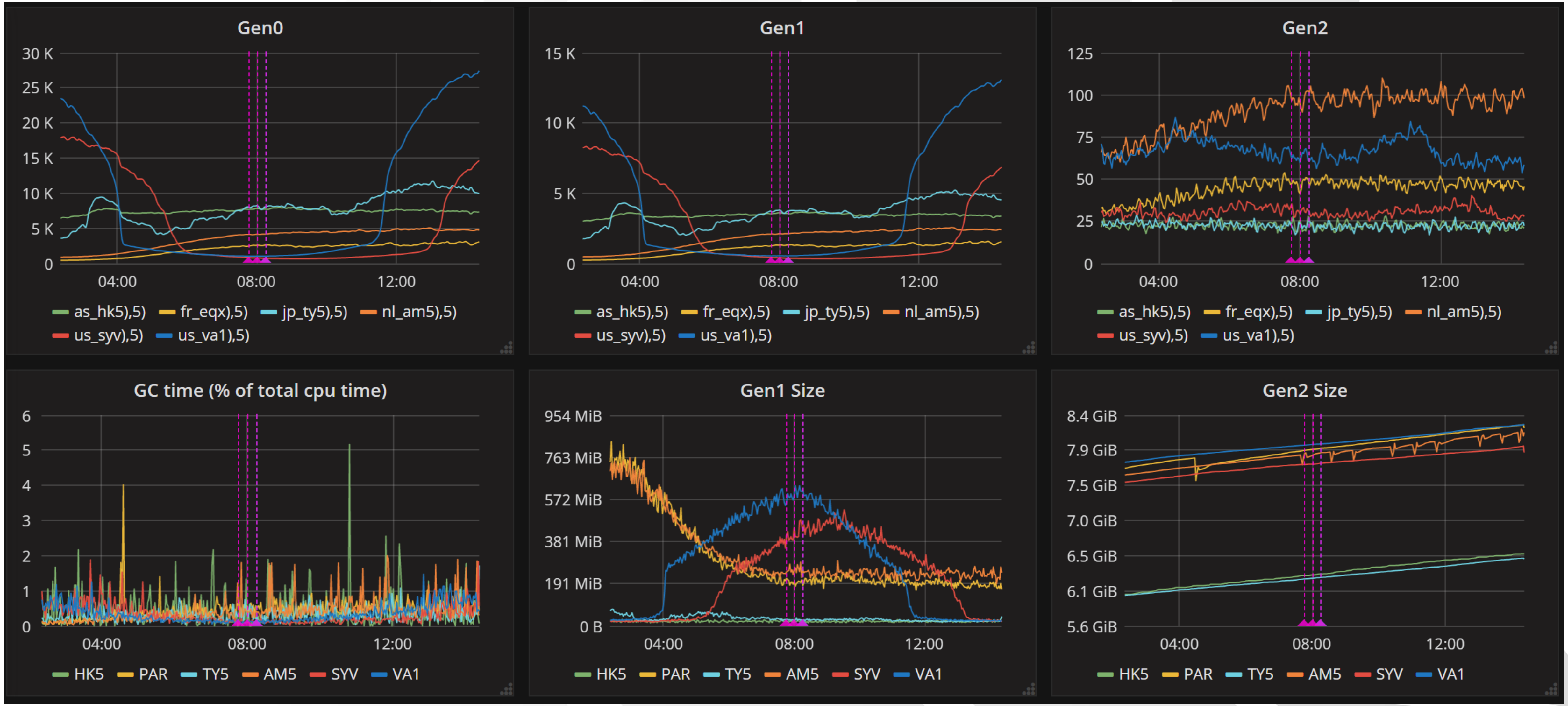
criteo

.NET Core Monitoring

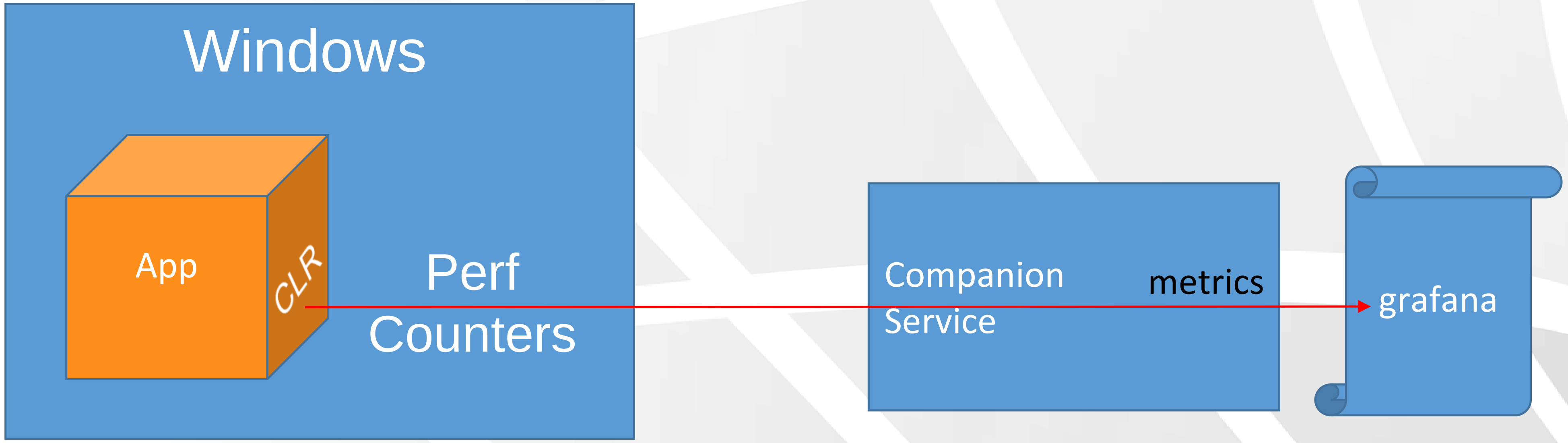
Criteo in numbers

- 9000+ servers running Windows
- 4000+ front-end servers
- 200+ billions requests per day

Currently on .NET 4.6.2 and... moving to .NET Core and Linux


Gen0
30 K
25 K
20 K
15 K
10 K
5 K
0
04:00
08:00
12:00
as_hk5),5)
fr_eqx),5)
jp_ty5),5)
nl_am5),5)
us_syv),5)
us_va1),5)
Gen1
15 K
10 K
5 K
0
04:00
08:00
12:00
as_hk5),5)
fr_eqx),5)
jp_ty5),5)
nl_am5),5)
us_syv),5)
us_va1),5)
Gen2
125
100
75
50
25
0
04:00
08:00
12:00
as_hk5),5)
fr_eqx),5)
jp_ty5),5)
nl_am5),5)
us_syv),5)
us_va1),5)
GC time (% of total cpu time)
6
5
4
3
2
1
0
04:00
08:00
12:00
HK5
PAR
TY5
AM5
SYV
VA1
Gen1 Size
954 MiB
763 MiB
572 MiB
381 MiB
191 MiB
0 B
04:00
08:00
12:00
HK5
PAR
TY5
AM5
SYV
VA1
Gen2 Size
8.4 GiB
7.9 GiB
7.5 GiB
7.0 GiB
6.5 GiB
6.1 GiB
5.6 GiB
04:00
08:00
12:00
HK5
PAR
TY5
AM5
SYV
VA1

Performance counters pipeline to Grafana



Performance counters limitations

- Misleading counters
  - Gen0 size
  - Gen collections count
  - Threads count
- Missing information
  - Suspension/contention time
  - Exceptions/finalizers
- **No perf counters in .NET Core**
  - Better than being blind…

Alternative to Perf Counters: CLR traces

- On Windows, Perfview has been using ETW for years

- Send to ETW on Windows and to LTTng on Linux
  - Stubs generated in CLR code at compile time via scripts\Python files

- Shared method that produces traces
  - Look for **`FireEtw<xxx>`** and **`ETW::<type>::<methods>`** functions
  - Great to find the undocumented events ;^)

Monitor Memory

- **Garbage Collections**
  - *GCHeapStats*
    - All generations size + pinned objects + promoted sizes
- **Application Threads Suspension Time during GC**
  - *GCSuspendEEStart/GCRestartEEStop*
- **Sampling heap allocations**
  - *AllocationTick*
    - Type name
    - Allocation size since the last event
    - LOH/SOH

Monitor Threading

- **Threads**
  - *ThreadCreating/ThreadRunning*
    - No way to know when a thread ends!
- **ThreadPool Worker threads**
  - *ThreadPoolWorkerThreadStart / ThreadPoolWorkerThreadStop*
    - Active/retired count
- **ThreadPool I/O threads**
  - *IOThreadCreationStart / IOThreadCreationStop*
    - Active/retired count

Monitoring/Investigation bonus!

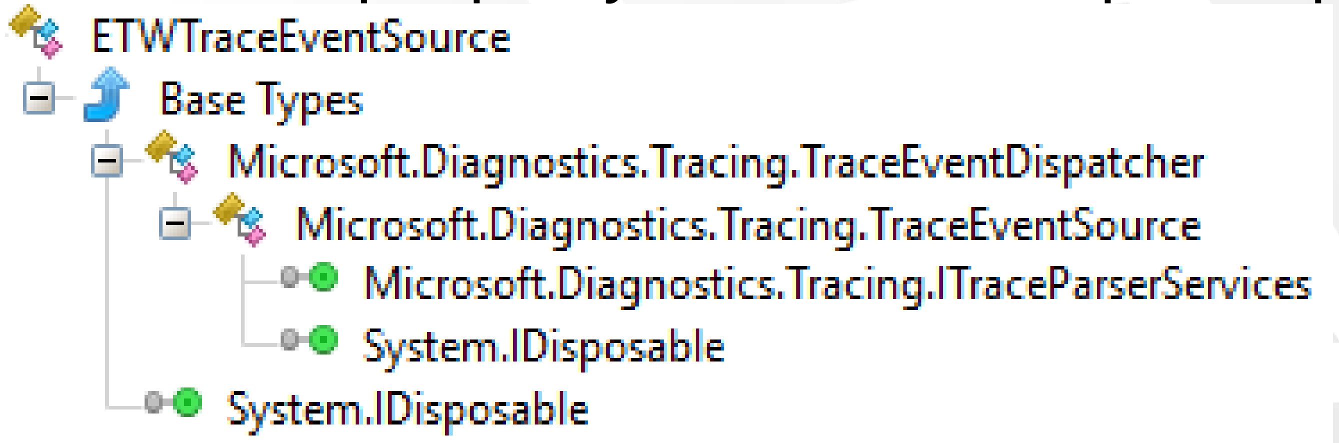
- **Exceptions Thrown**
  - *ExceptionStart*
    - Exception type
    - Exception message
- **Finalizers**
  - *GCFinalizeObject/GCFinalizersStart/GCFinalizersEnd*
    - Type name of the finalized instance
    - Count and time spent
- **Contention time**
  - *ContentionStart/ContentionStop*

Connecting to CLR with TraceEvent (Windows)

- Creating a `TraceEventSession`
  - Give it a unique name
  - Don’t forget to dispose it
    - `logman -ets stop <session name>`
- Enable `ClrTraceEventParser.ProviderGuid`
  - Pick the right event level (depending on events)
  - Filter events by keywords
  - Additional options set via `TraceEventProviderOptions`

Decyphering CLR traces with TraceEvent

- The `Source` property on session exposes parsers



```
ETWTraceEventSource
  Base Types
    Microsoft.Diagnostics.Tracing.TraceEventDispatcher
      Microsoft.Diagnostics.Tracing.TraceEventSource
        Microsoft.Diagnostics.Tracing.ITraceParserServices
        System.IDisposable
    System.IDisposable
```

  - Use the parser exposed by the source `Clr` property
    - The `All` event is raised each time a trace is received
- Start the processing of events thanks to `Process()`
  - This is a blocking call!

Naive usage of TraceEvent

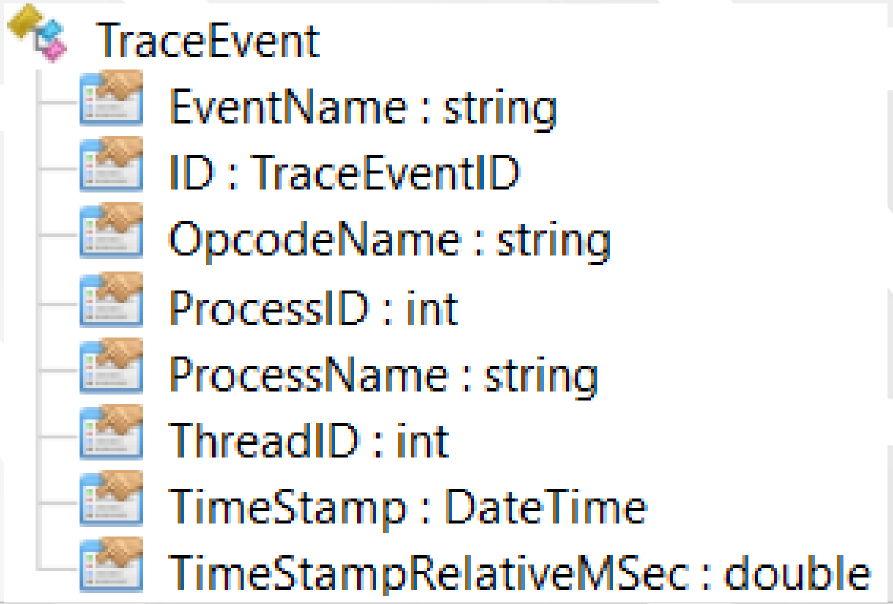
Demo

- Notion of session
- Setup providers
- Listen to All events

Decyphering CLR events with TraceEvent

- Listen to Clr parser events
  - Dig into the documentation for details

- Each event has a common `TraceEvent` payload



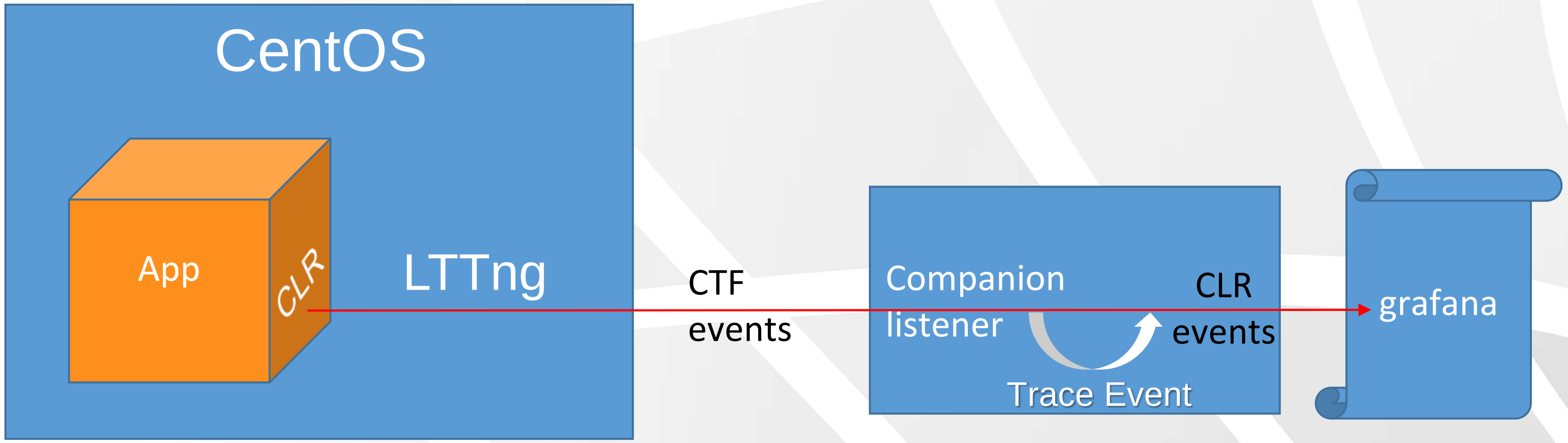
```
TraceEvent
  EventName : string
  ID : TraceEventID
  OpcodeName : string
  ProcessID : int
  ProcessName : string
  ThreadID : int
  TimeStamp : DateTime
  TimeStampRelativeMSec : double
```

Decyphering CLR events with TraceEvent

Demo

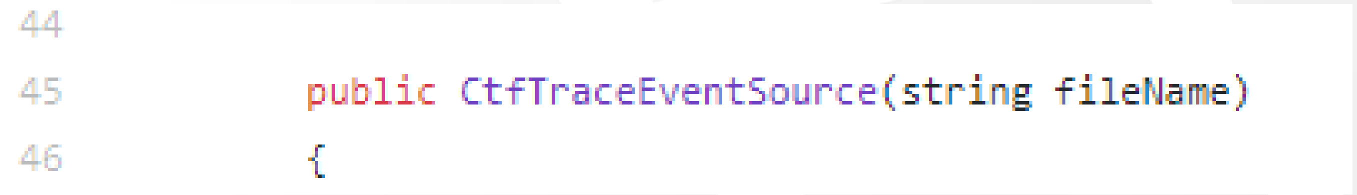
- First chance exception: unique event
- Thread contention: start/stop events

CentOS CLR events pipeline to Grafana



Decyphering CLR event produced as CTF traces

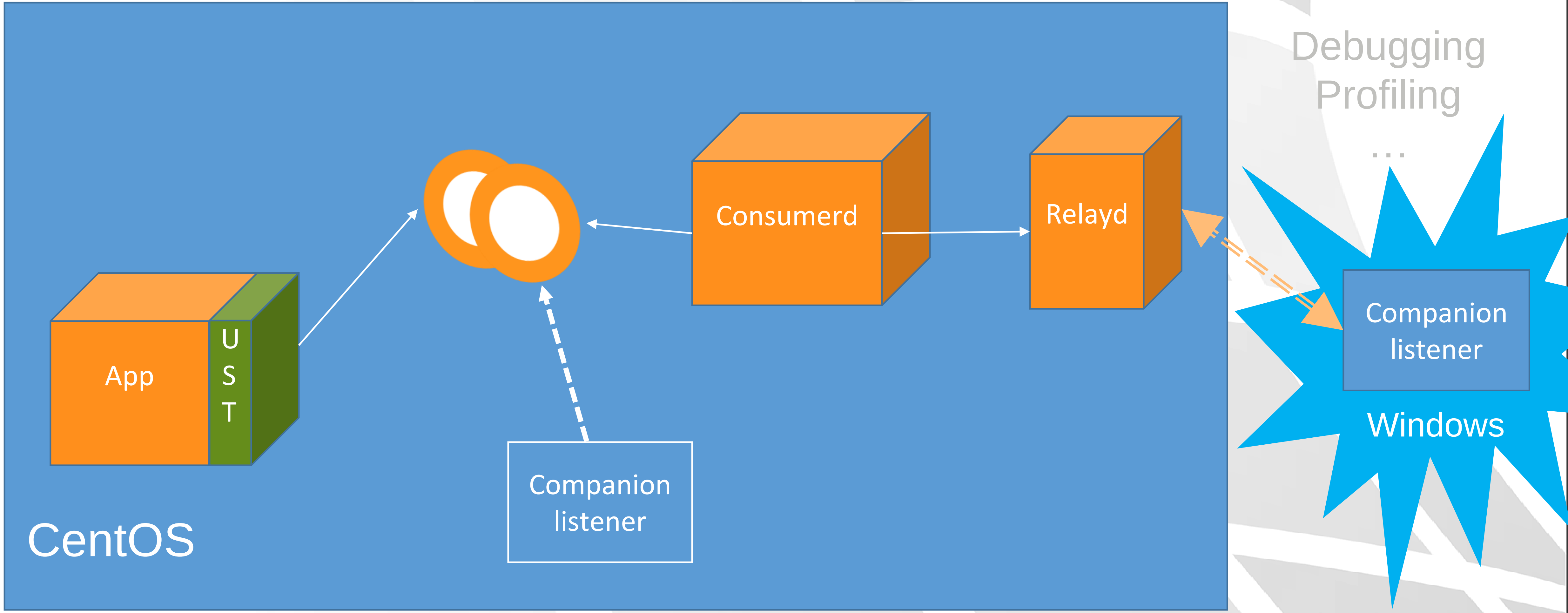
- Use TraceEvent library but… only file-based ctor



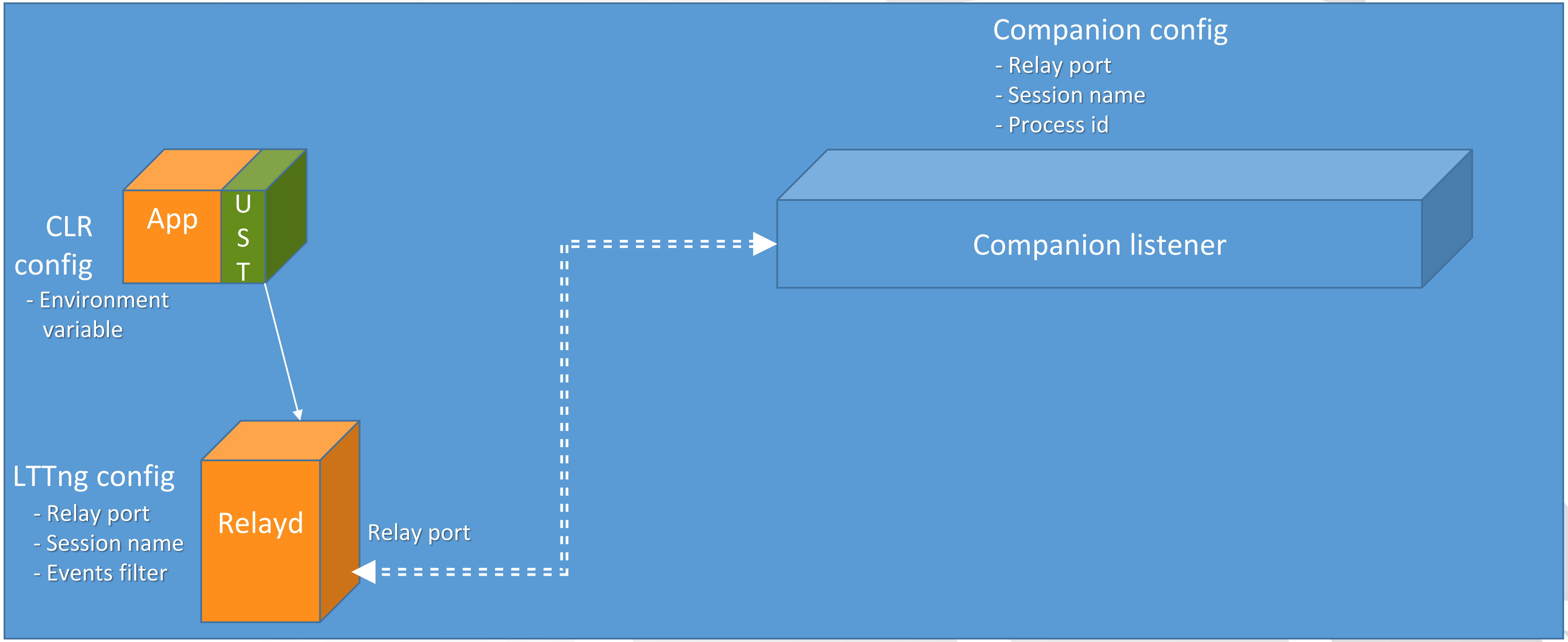
```
        public CtfTraceEventSource(string fileName)
        {
```

- Implement CTF live session support:
  https://github.com/Microsoft/perfview/pull/340

How to integrate the LTTng pipeline in Criteo



LTTng Live Session implementation details



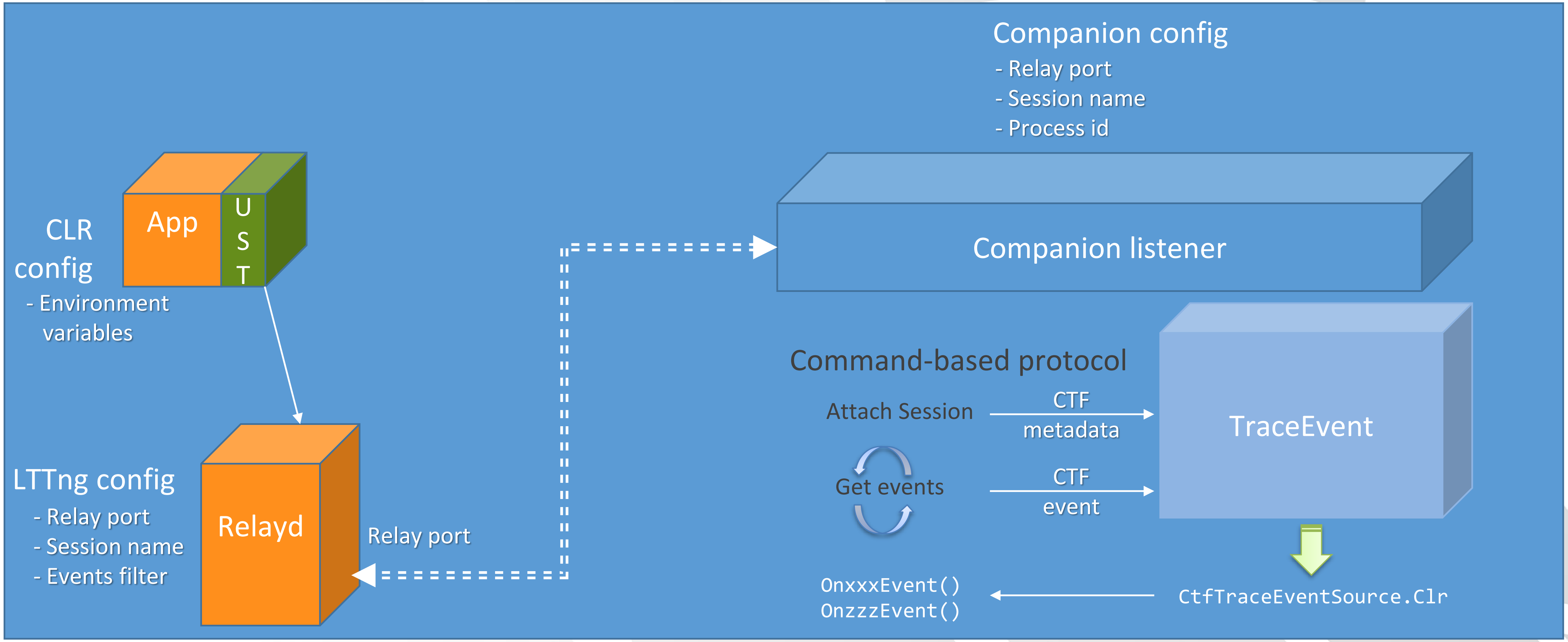
Configuring pipeline on Linux

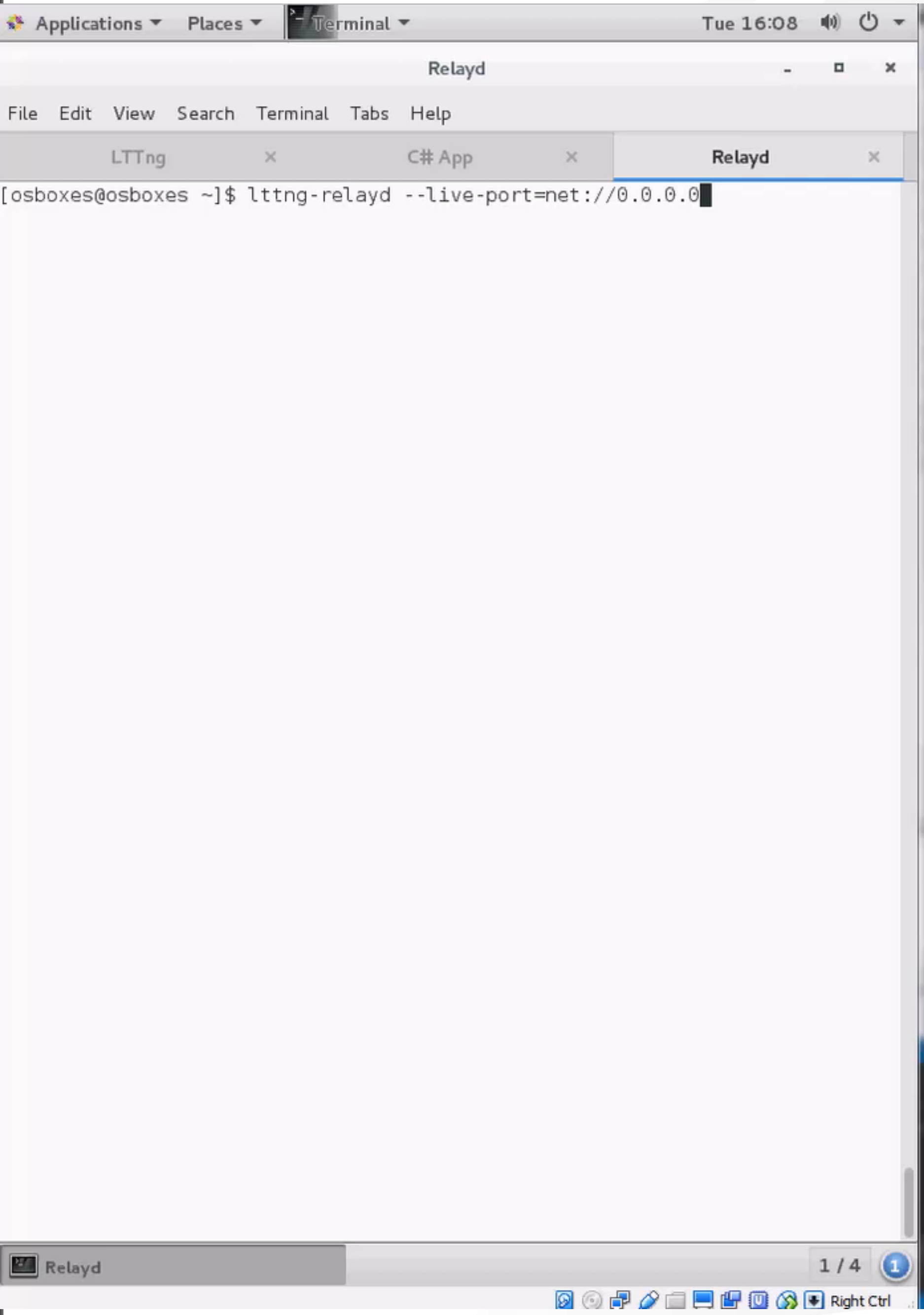
- LTTng configuration

```
lttng-relayd -d --control-port=tcp://0.0.0.0:8080
--data-port=tcp://0.0.0.0:8081 --live-port=tcp://0.0.0.0:8088

lttng create sela_conference_session --live=100000
--ctrl-url=tcp://0.0.0.0:8080 --data-url=tcp://0.0.0.0:8081

lttng enable-event --userspace -c sela_conference_session
--tracepoint DotNETRuntime:*
```

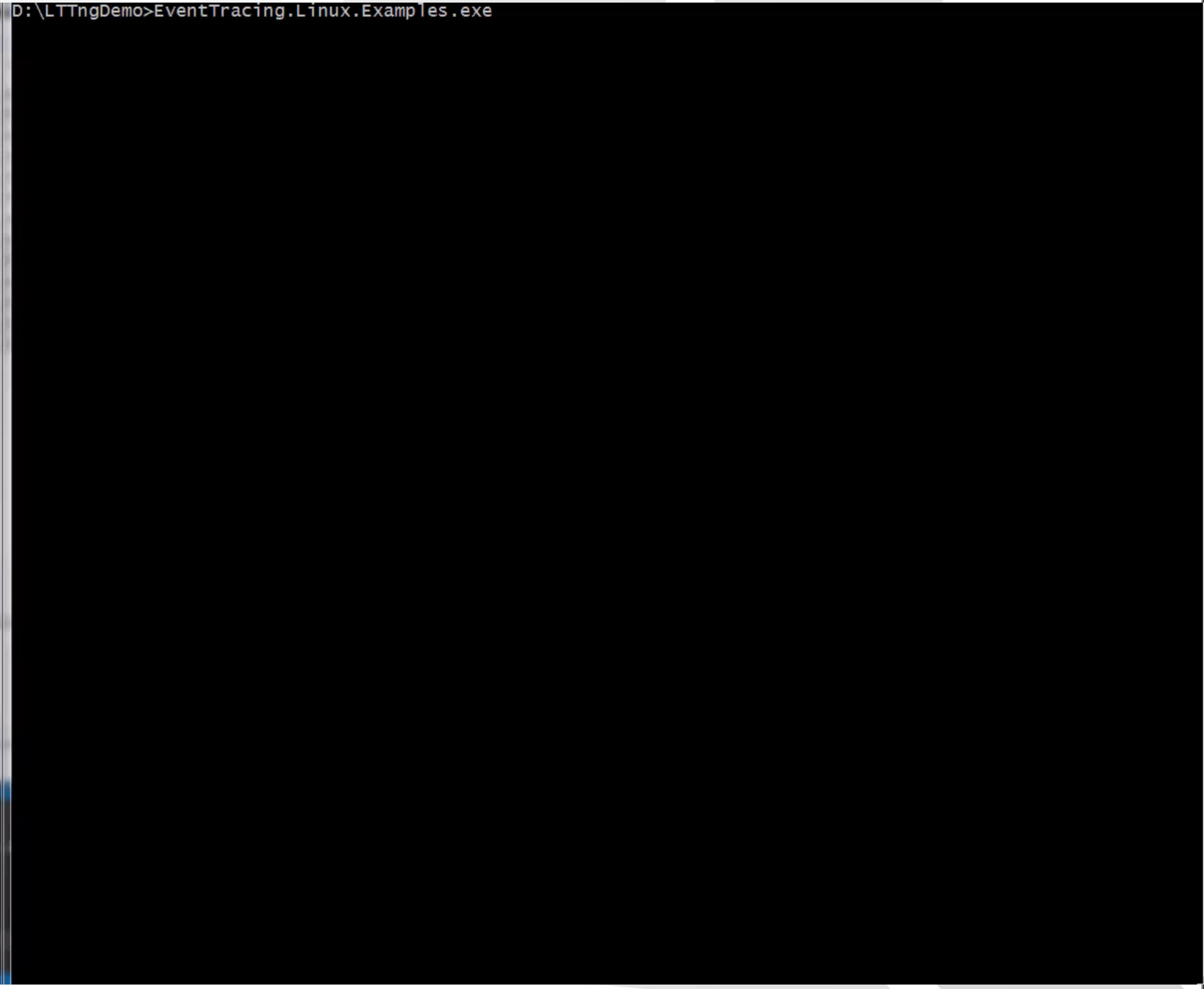
- Environment variables for CLR
  - `export COMPlus_PerfMapEnabled=1`
  - `export COMPlus_EnableEventLog=1`

LTTng Live Session implementation details





```
[osboxes@osboxes ~]$ lttng-relayd --live-port=net://0.0.0.0
```



```
D:\LTTngDemo>EventTracing.Linux.Examples.exe
```

LTTng pitfalls

- The relay is also storing events on disk
  - Customizable by log rotation
- LTTng seems to have a lower throughput than ETW
  - Filter your CLR events carefully in LTTng configuration (ex: AllocationTick)

Key takeaways

- Prefer CLR traces to performance counters
- No alternative when moving to .NET Core
- Learning curve moving to Linux could be…

Blog series about performance counters and ETW:
http://labs.criteo.com/2018/06/replace-net-performance-counters-by-clr-event-tracing/

Questions