Silexica Facts

- Est. 2014 after a decade of research
- Team of world leading software and hardware experts
- 60 people worldwide, engineering HQ in Germany
- 3 offices and worldwide local support engineers
Motivation

- Autoware ROS-based AD stack (Moriyama example, v1.8)
- Problem: Positioning of the car is sometimes unstable
- Many active modules in the example + large codebase:
  - Where to start?!
  - How to make behavior reproducible!?
Motivation

Two Runs of the Same Scenario

Successful run:
- Positioning module (marked red) runs in regular intervals with acceptable runtime variation

Unsuccessful run:
- Positioning module (marked red) does not run in regular time intervals and has a highly variable runtime
Problem Statement

Missing system overview → Engineers are fixing, not developing

No statistical profiling of system → High risk of (creeping) system degradation

No automated testing of system metrics → Last-minute fire-fighting / delay of release

No scalability of testing solution → Limited test coverage to achieve high release rate
SLX Analytics & Testing Platform

- Automated outlier detection:
  - Multi-level/run tracing + analysis
  - CI integration

- Interactive root cause analysis

- Highly scalable:
  - Cloud, on-premise, desktop
  - Open API
Multi-Level Analysis

- **System-level:**
  - Acquire kernel/system events → global timeline of the system

- **Middleware-level (e.g. ROS{1, 2}, Adaptive AUTOSAR, etc.):**
  - Acquire middleware-specific data (e.g. from the communication stack) to provide semantic context

- **Module-level:**
  - Acquire module-specific data (static/dynamic) for in-depth analysis
Multi-Level Analysis

System-level data: Global CPU timeline (processes)
Multi-Level Analysis

Use middleware-level data to highlight relevant processes
Multi-Level Analysis

Add middleware communication events to global timeline
Multi-Run Analysis

- Run test scenarios multiple times to detect outliers
- Configurable constraints for automated tests
- Full results history:
  - Compare code revisions
  - Compare same/different inputs
Module-Level Analysis

- Detect memory allocations after startup phase
- Detect system calls with difficult to predict runtime behavior
- Detect TCP, UDP, and middleware communication

File I/O:
- Read: 12 kB
- Write: 0 kB

TCP/IP:
- Read: 3 kB
- Write: 4 kB

Memory allocated: 10 MB
Stress Tests

- Highly configurable stress tests: CPU, memory, cache, etc.
- Loadable/storable stress profiles (stress over time)
- Chaos engineering: Randomly stop processes during execution
Interactive Root Cause Analysis

Latency Test History

Constraint violated! Investigate (click!)

Constraint: 850 ms

Commits

Measured latency distributions

Level 1
Interactive Root Cause Analysis

Latency Histogram

Constraint: 850 ms

Measured latencies

Investigate further (click!)
Interactive Root Cause Analysis

Latency Over Time

Constraint: 850 ms

Investigate further (click!)

Measured latency over time
Interactive Root Cause Analysis

CPU Gantt Chart

Processes

System state at time of constraint violation

Level 4

CPU s

Time / s
Open API / Jupyter Notebooks

```python
import slx.api
engine = slx.api.db engine()
slx.api.init_plots()

config_aarch64 = {
    'arch': 'aarch64',
    'OS': 'Ubuntu 18.04 bionic',
    'scenario': 'perception',
    'tag': None
}

exp_ids_aarch64, exp_configs_aarch64, exp_infos_aarch64 = slx.cmd.list_experiments(engine, config_aarch64)

# Create a plot for the given metric
plot_cfg = slx.api.make_plot_config_experiments(slx.api.Metric.CPU_LOAD_BY_CPU, hist_avg=False, line_all=True)
slx.api.plot_experiments(engine, plot_cfg, exp_ids_aarch64, exp_configs_aarch64)
```

Select Results

Select Metric

Visualize
Thanks

1st product release in January 2020