Analysing & Solving Qt UI Performance Problems On Embedded Hardware

Phil Brumby,

Mentor Graphics

Agenda

- Introduction
- Performance Matters!
- Typical performance problems
- Methods of solving performance problems
- Examples of using trace analysis
- Conclusion/Questions

Introduction

About Me

- Phil Brumby, Senior Technical Marketing Engineer
- Former software engineer in user interface technology and mobile gaming space
- Specialised in Graphics & UI at Mentor Embedded
  - Working closely with our embedded customers & their needs
  - Across OS's including Nucleus (Mentors own RTOS), but also Embedded Linux & Android
- Focused and valued approach on this topic in that I come at it from the end users perspective!

Mentor Embedded

400+ engineers

50+ engineers in lead OSS community roles

10,000+ accepted OSS changes

Software deployed in over 3 Billion devices

Over 20,000 users of embedded tools

www.mentor.com/embedded

Mentor Embedded & Qt

- 20+ years of experience working in the embedded market, enabling customers in automotive, industrial, medical devices, and consumer electronics
  - Commercially supported and customizable Linux®
  - For real-time control systems developers can take advantage Nucleus® RTOS.
- Mentor has been working to enable the use of Qt with its own portfolio of embedded software & tools:
  - Integration of Qt run-time on our own RTOS Nucleus
  - Port covers Qt Core & GUI for QWidget solutions
  - Genivi compliant Linux Automotive Technology Platform (ATP)

Performance Matters!

Why does UI performance matter to Mentor Embedded?

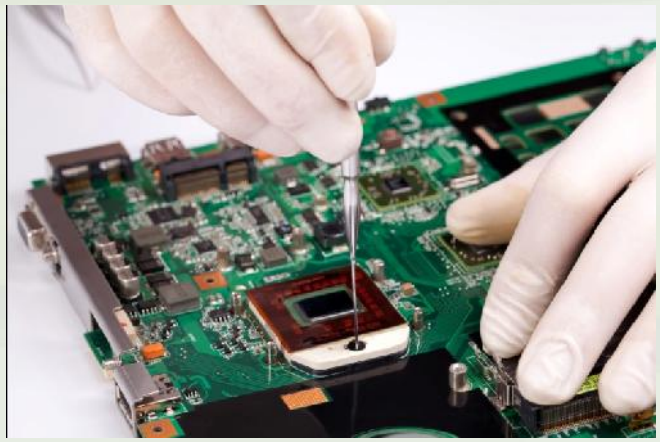
"No matter how good your underlying system is, the users will only remember your user interface. Fail there and you will fail, period." -- Tristan Louis

- For Mentor, after enabling a customer with UI technology, OS, Middleware and Dev tools, we have a vested interest in ensuring a quality product is produced!

Why are performance problems an issue today?

- Expectations from the ‘Smart Phone’ experience has raised consumer expectations, significantly raising the performance bar for all embedded devices.

- Despite advancements in hardware capability embedded UI development must still pay careful attention to platform capabilities!



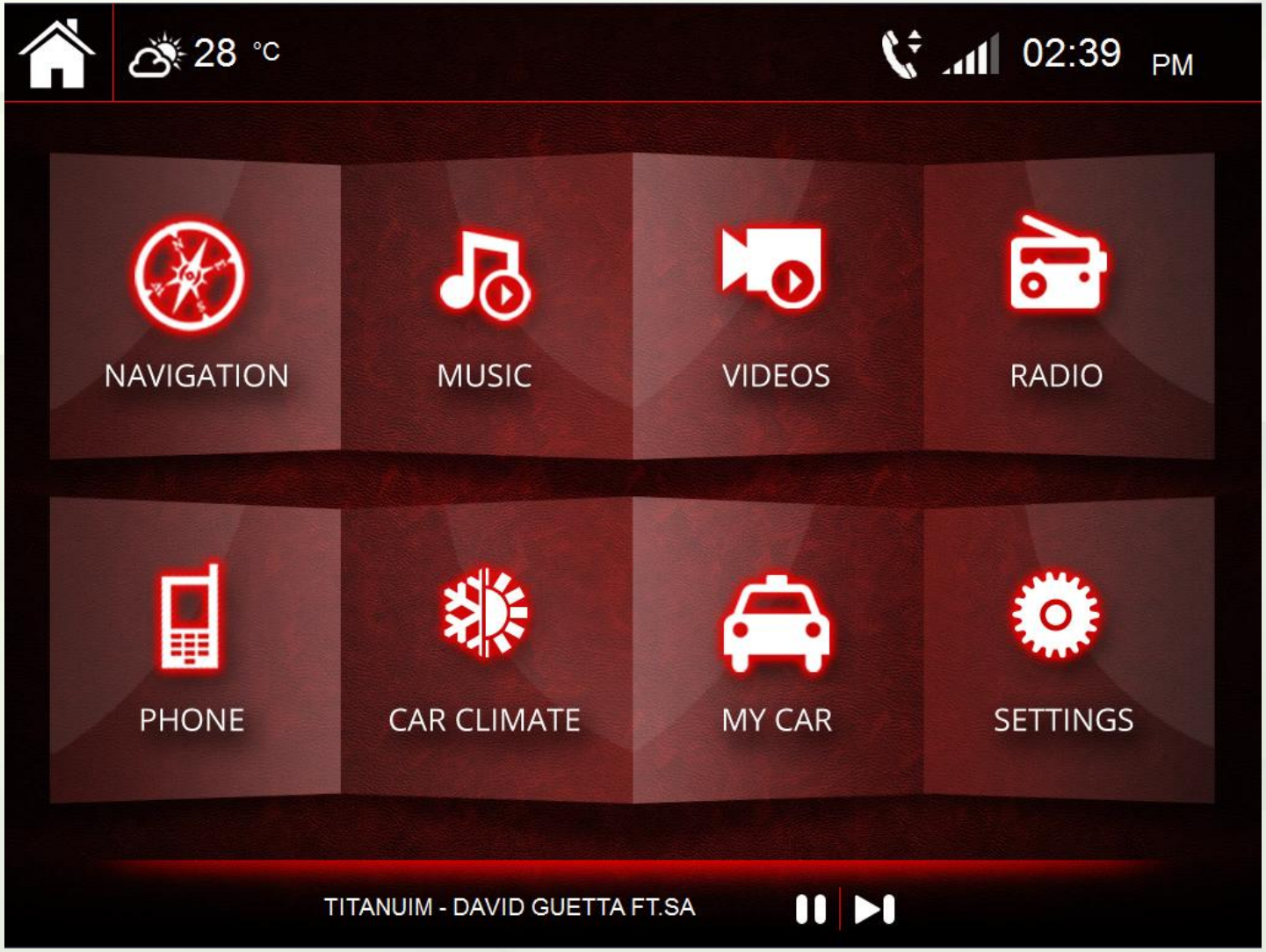
Typical Performance Problems

Typical Performance Problems

- Responsiveness
  - How quickly a UI responds to an input (e.g. a touch event)
  - Avoiding UIs which are perceived as “laggy”
- Animation smoothness
  - Typically measured in frames per second
- Start-up time
  - Affected by OS, UI framework, application processing, resource loading and graphics computation

Example – Bad user interaction experience

In Vehicle Infotainment UI, QML Design, Linux, iMX6 board



Performance Metrics

- To get to the bottom of performance issues, we must understand system metrics for measuring performance
- We can then use them to tell the story of just what is happening on the hardware at any given time
  - Execution profile – LISRS, HISRS, Tasks, Events
  - CPU – State, utilisation, multi core access
  - Memory – Pools, usage, load
  - File system activity
- These should be coupled with computational user defined metrics
  - Frame rate (FPS), pixel load, runtime data requests, % screen redraw, start up time…

A QWidget or QML solution?

- From engagements experienced in the embedded space significant user cases exist for deployment of both Qt solutions.
- We still see the need to deploy QWidget UIs
  - Memory considerations - smaller footprint
  - No GPU
  - Some markets still require static, traditional 2D buttons and controls style UIs - Medical
- Ultimately the best performance analysis tools should cover both!

Methods/Tools to Analyse Performance

Printf

- “gdb didn't help, so I'll add some printfs.”
- Usage:
  - `printf(“REDRAW@%ld, foo=%x\n”, gettimeofday(stuff), foo);`
- Pros
  - Simple & easy; works (almost) everywhere
- Cons
  - Low performance, inefficient
  - Add; rebuild; re-run; tweak; rebuild; re-run; remove; rebuild; re-run; …
  - **Post-processing the results**
- Right tool for **displaying output**; wrong tool for this job

```
REDRAW@87234564, foo=a
REDRAW@87234456, foo=3
!!! media file '/usr/share/myapp/media/bar.png' changed; reloading !!!
REDRAW@87234576, foo=8
click_down 27,432
REDRAW@87234576, foo=8
click_up
redrawRegion @87234698: 0,100-230x80
```

Statistical Profiling

- Determine which code is *typically* the greatest user of CPU or cache.
- Usage:
  - perf record mytestcase; examine table for highest consumers
- Pros
  - High performance
- Cons
  - Focused on utilization of *hardware* resources
  - Static table aggregating all results; can't narrow focus to problems
  - **Only helps to visualize repeated patterns**
- May be right tool for **throughput problems**; wrong tool for other jobs.

```
12.95% ls ls [.] 0x00004a3e
7.18% ls libc-2.17.so [.] 0x0007e757
6.03% ls libc-2.17.so [.] __strcoll_l
3.37% ls [kernel.kallsyms] [k] __ticket_spin_unlock
2.63% ls [kernel.kallsyms] [k] __ticket_spin_lock
2.22% ls [kernel.kallsyms] [k] n_tty_write
1.94% ls [kernel.kallsyms] [k] _raw_spin_unlock_irqrestore
1.69% ls [kernel.kallsyms] [k] memset
...
```

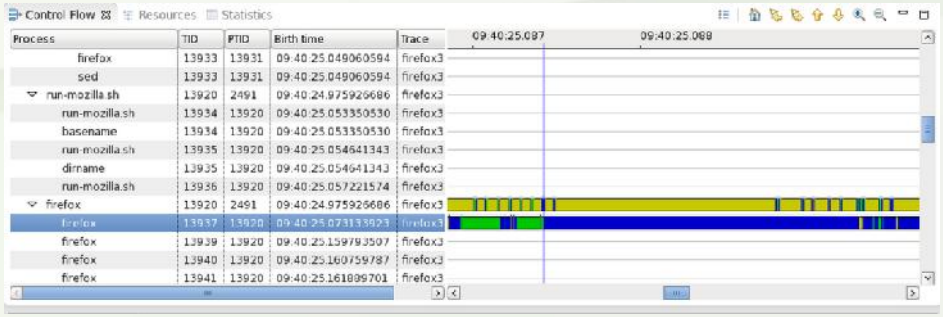
Tracing

- Like printf but lower level and higher-performance
- Usage:
  - tracepoint(REDRAW, foo);
- Pros
  - High performance
  - Correlates activity at many layers
- Cons
  - **Post-processing the results**

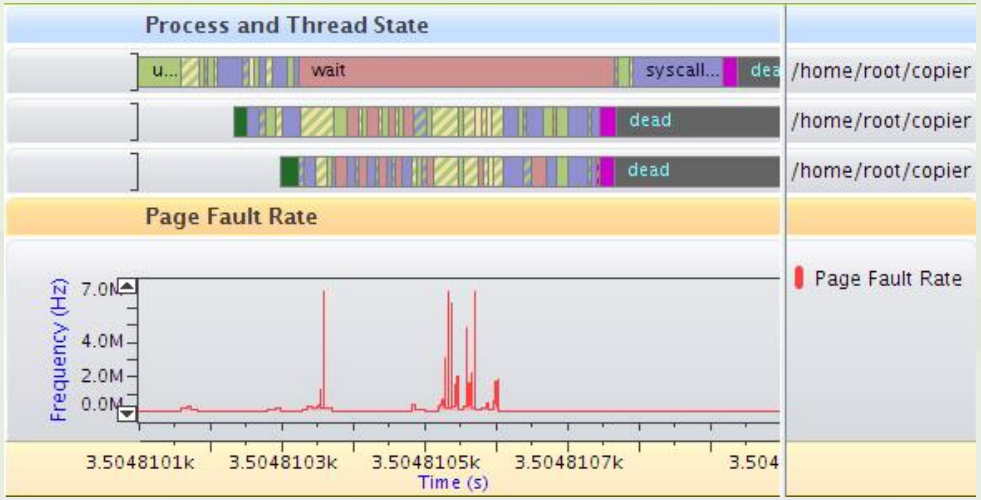
```
[05:19:37.099741586] sys_geteuid: { cpu_id = 4 }, { }
[05:19:37.099742180] exit_syscall: { cpu_id = 4 }, { ret = 0 }
[05:19:37.099743115] sys_pipe: { cpu_id = 4 }, { fildes = 0x7FBA4D454AD0 }
[05:19:37.099750095] exit_syscall: { cpu_id = 4 }, { ret = 0 }
[05:19:37.099751030] sys_mmap: { cpu_id = 4 }, { addr = 0x0, len = 10485760, prot = 3, flags = 131362,
    fd = -1,
offset = 0 }
[05:19:37.099755483] exit_syscall: { cpu_id = 4 }, { ret = 140438029725696 }
[05:19:37.099960298] timer_init: { cpu_id = 7 }, { timer = 18446612138716355504 }
[05:19:37.099961822] timer_start: { cpu_id = 7 }, { timer = 18446612138716355504, function =
    18446744071579164944,
expires = 4311019744, now = 4311009744 }
```

Tracing Viewing

- We have a haystack of data now, but where is the needle?
- Pros
  - Can show the events graphically
  - Basic features like search & filter
- Cons
  - Depend on users to find the patterns
  - Fixed or limited data sets, no custom application level viewing

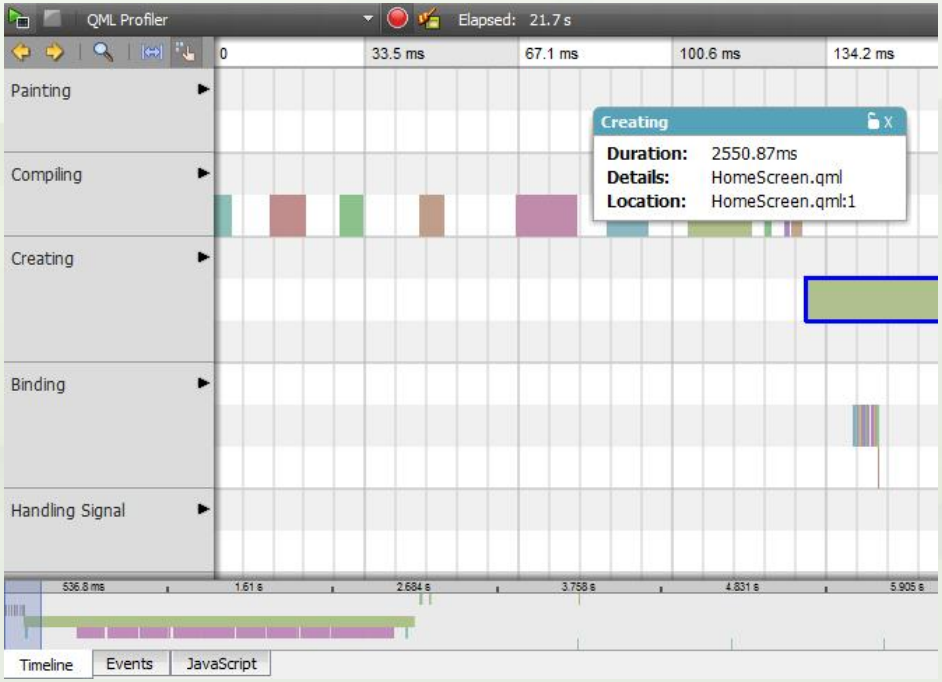


- Need a more flexible means of analysis to calculate and display to the user higher-level patterns of data



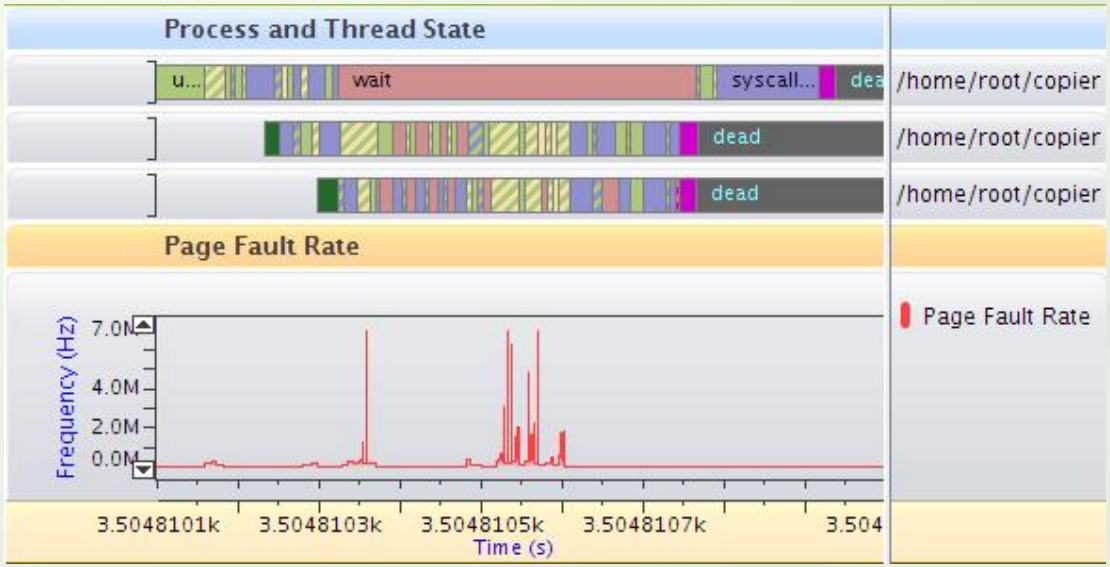
QML Profiler

- Purpose-optimized performance tool for QML.
- Usage:
  - “I'm using QML and I think my problem is strictly within my application.”
- Pros
  - Deep QML comprehension
- Cons
  - No system-wide interactions
  - No QWidget coverage



Trace Analysis

- Task-centric analysis to calculate and display a system wide and more user defined visual analysis of the system

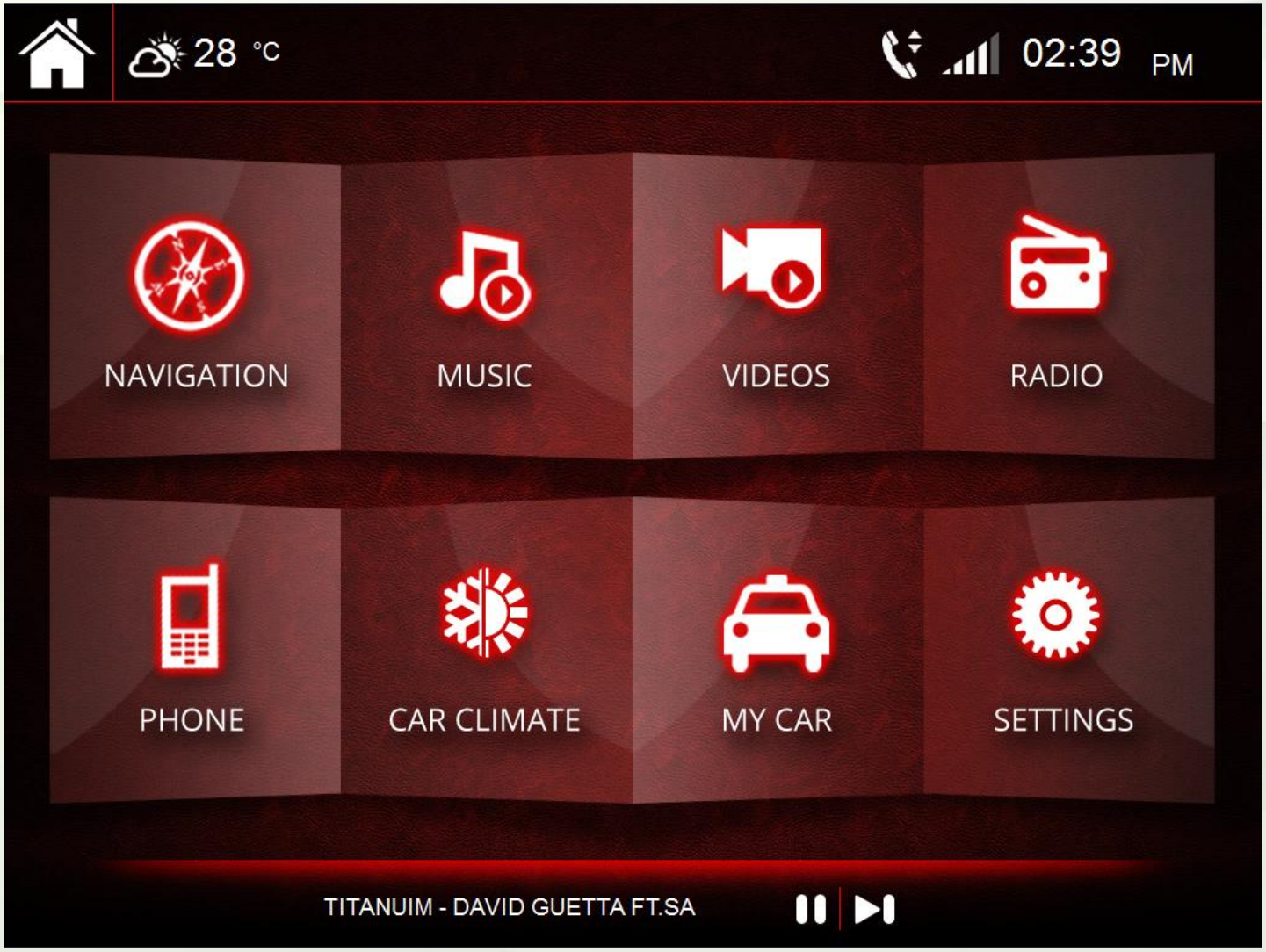


- Live demo…

Examples of Using Trace Analysis

Use Case 1 – Bad user interaction experience

In Vehicle Infotainment UI, QML Design, Linux, iMX6 board



Use Case 2 – Intermittent framerate issues

SEP 2.0 Washing Machine UI, Qt Widget, Nucleus RTOS, Qemu

Conclusions

Conclusion

- There are lots of tools out there. Using the right one for each job makes all the difference.
- Tracing is a good approach to many system wide performance problems, but needs a tool to process & help visualise it all.
- Sourcery Analyzer can perform trace analysis at the OS layer, the Qt layer, and even the application layer.
- http://go.mentor.com/sourceryanalyzer
- Questions?