DEVELOPER DAYS 2014 EUROPE

Analyzing Performance of QtQuick Applications

Thomas McGuire
KDAB

thomas@kdab.com

Performance: Multiple Aspects

- **Startup Duration**
- **Smooth Rendering / Frames per Second**
- Responsiveness
- Boot Duration
- Power Usage
- Memory Usage

Startup Time

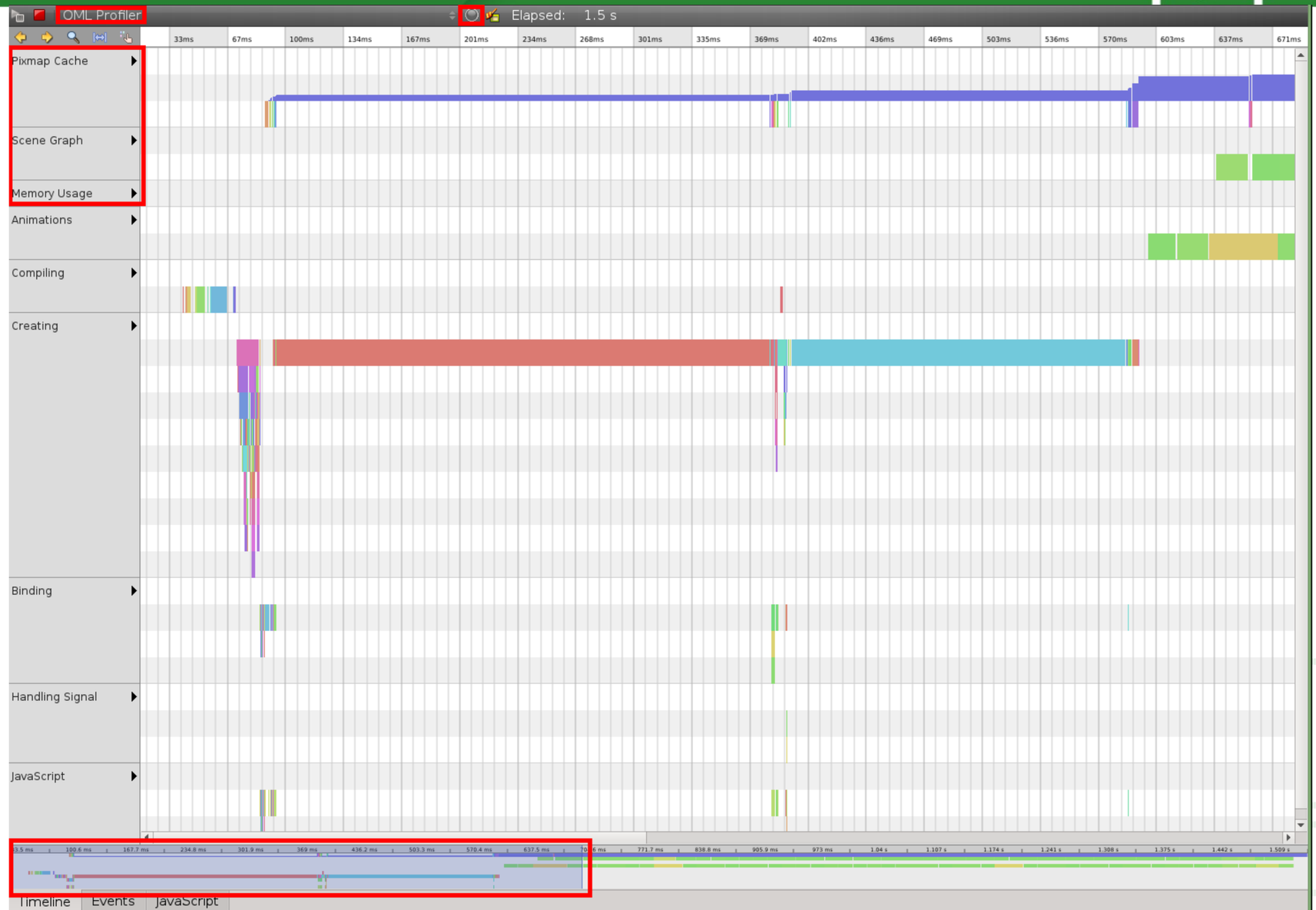
Startup Time - CPU Profiler

Function Stack	CPU Time: Total by Utilization (Idle, Poor, Ok, Ideal, Over)
Total	634.645ms
__libc_start_main	634.645ms
main	634.645ms
QQmlEngine::rootContext	0ms
QQmlContext::setContextProperty	1.772ms
Storage::Storage	5.316ms
[redacted]Storage::[redacted]Storage	7.210ms
loadFonts	39.709ms
QGuiApplication::QGuiApplication	48.122ms
MainView::MainView	81.629ms
MainView::setMainQmlFile	412.104ms
QQuickView::setSource	412.104ms
QQuickViewPrivate::execute	412.104ms
QQmlComponent::QQmlComponent	0.595ms
QQuickView::continueExecute	411.509ms
v8::internal::CallIC_Miss	0ms
v8::internal::LoadIC_Miss	1.771ms
QQmlComponent::create	409.738ms
QQmlComponent::beginCreate	44.360ms
QQmlComponentPrivate::beginCreate	44.360ms
QQmlComponent::completeCreate	365.378ms

Startup Time - CPU Profiler

- Pay attention to what you measure
  - Cycle count does not include time blocked!
  - Compile in release mode
  - Profile on target device
  - Profile with cold cache
- User code and QML engine code
  - QML engine part opaque
  - **high level tooling required**

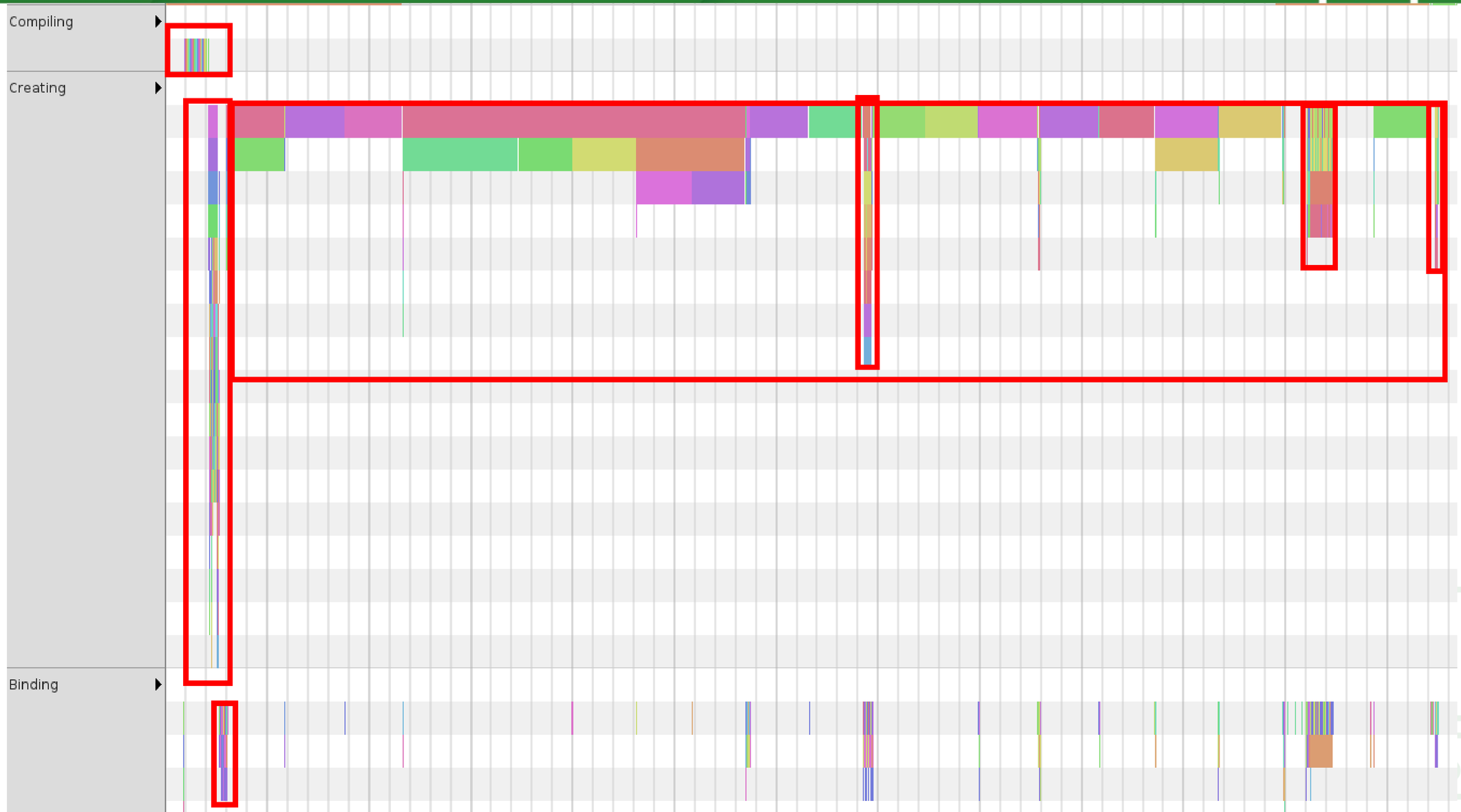
Startup Time - Meet the QML Profiler



Startup Time - Meet the QML Profiler

- Use Qt 5.4 and QtCreator 3.2
- Enable profiler in settings
  - QMake *CONFIG* flag
  - *run* argument
- Record only what you need

Startup Time - Example



Startup Time - 4 phases

1. **Compiling**
2. **Creating**
3. **Bindings**
4. **Completion**
   - JS: *Component.onCompleted*
   - C++: *QQuickItem::componentComplete()*
   - Text layouting, image loading, creation of Repeater/ListView delegates, ...

Startup Time - Completion

QQuickText::componentComplete	0.862s
QQuickTextPrivate::ensureDoc	0.001s
QQuickItem::componentComplete	0.205s
QQuickTextPrivate::updateLayout	0.656s
QQuickStyledText::parse	0.000s
v8::internal::CompareIC_Miss	0.001s
v8::internal::LoadPropertyWithInterceptorForLoad	0.001s
QQuickTextPrivate::updateSize	0.655s
QTextDocument::setPageSize	0s
QQuickText::contentSizeChanged	0.001s
QFontMetricsF::height	0.098s
QQuickTextPrivate::setupTextLayout	0.556s
QQuickItem::setImplicitSize	0.000s
QTextLayout::beginLayout	0.001s
QQuickTextPrivate::setLineGeometry	0.555s
QTextLine::setLineWidth	0.555s
QTextLine::layout_helper	0.555s
QTextEngine::attributes	0.000s
QFontEngineMulti::minRightBearing	0.002s
QTextEngine::shape	0.553s
QTextEngine::shapeText	0.553s
QTextEngine::shapeTextWithHarfbuzz	0.553s
qShapeItem	0.001s
QTextEngine::fontEngine	0.552s
QFontPrivate::engineForScript	0.552s
QFontDatabase::load	0.552s
QFontDatabase::findFont	0.552s
match	0.001s
QFontDatabase::parseFontName	0.001s
loadEngine	0.551s
loadSingleEngine	0.551s
QFontconfigDatabase::fontEngine	0.551s

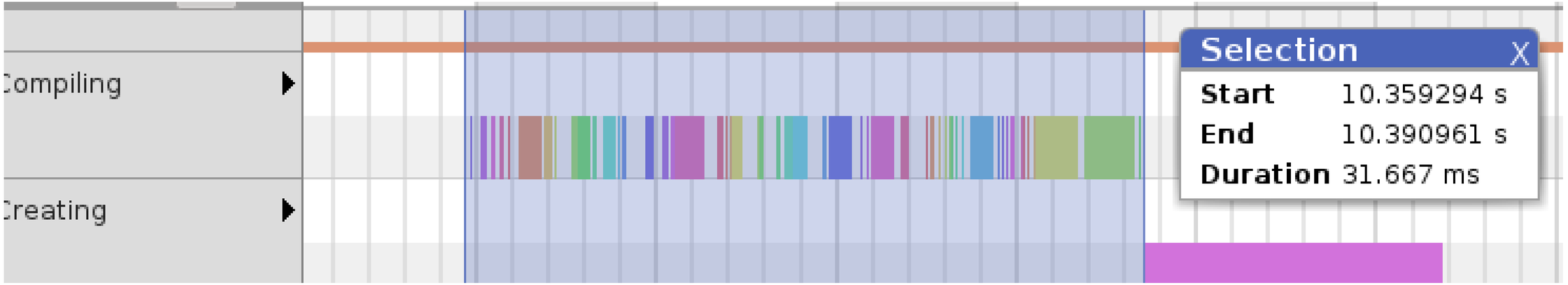
Startup Time - Completion

```
531     FcResult result;
532     FcPattern *match = FcFontMatch(0, pattern, &result);   550.828ms
533     if (match) {
```

```
thomas@witwewackelpudding:~:master: fc-list | wc -l
2924
```

- Removing fonts improved startup from 900ms to 200ms
- Completion phase shrunk considerably

Startup Time - Compilation



- Compilation phase fast, small amount of total
- Runs in a separate thread
- QtQuick Compiler pre-compiles files
  - Phase reduced by ~50%
  - Available since Qt 5.3 Enterprise

Startup Time - Bindings/JS

- Keep bindings simple
- Move complex code to C++
- Use QtQuick compiler if available

Startup Time - QtQuick Compiler

```
function stuff()
{
    console.time("Stuff");
    var a = Math.random(100);
    var b = Math.random(100);
    var c = Math.random(100);
    var sum = 0;
    for (var x = 0; x <= 10000; x++) {
        for (var i = 0; i <= 10000; i++) {
            var d = i*a + i*b + i*c;
            sum += d;
        }
    }
    console.timeEnd("Stuff");
    console.log("SUM= " + sum);
}
```

Startup Time - QtQuick Compiler

- Results
  - Without QtQuick Compiler, Release: 1000ms
  - With QtQuick Compiler, Release: 500ms, 398 instructions (w/o calls)
  - With QtQuick Compiler, Debug: 5000ms, 818 instructions (w/o calls)
  - C++ version, Release: 50 ms, 78 instructions (w/o calls)
- Use QtQuick Compiler if available
- Improvements in simpler code (bindings) ~15% (*)
- Move complex code to C++

Startup - Creating

- Not much one can do
- Use fewer elements in QML files
- Make sure custom items are constructed quickly

Startup - All phases

Use **Loader** to load views later

Startup - Summary

- Profile both C++ and QML
- Know your tools, understand their output
- Move complex JS code to C++
- Use Loaders
- Use QtQuick Compiler when available

Smooth Rendering / Frames per Second

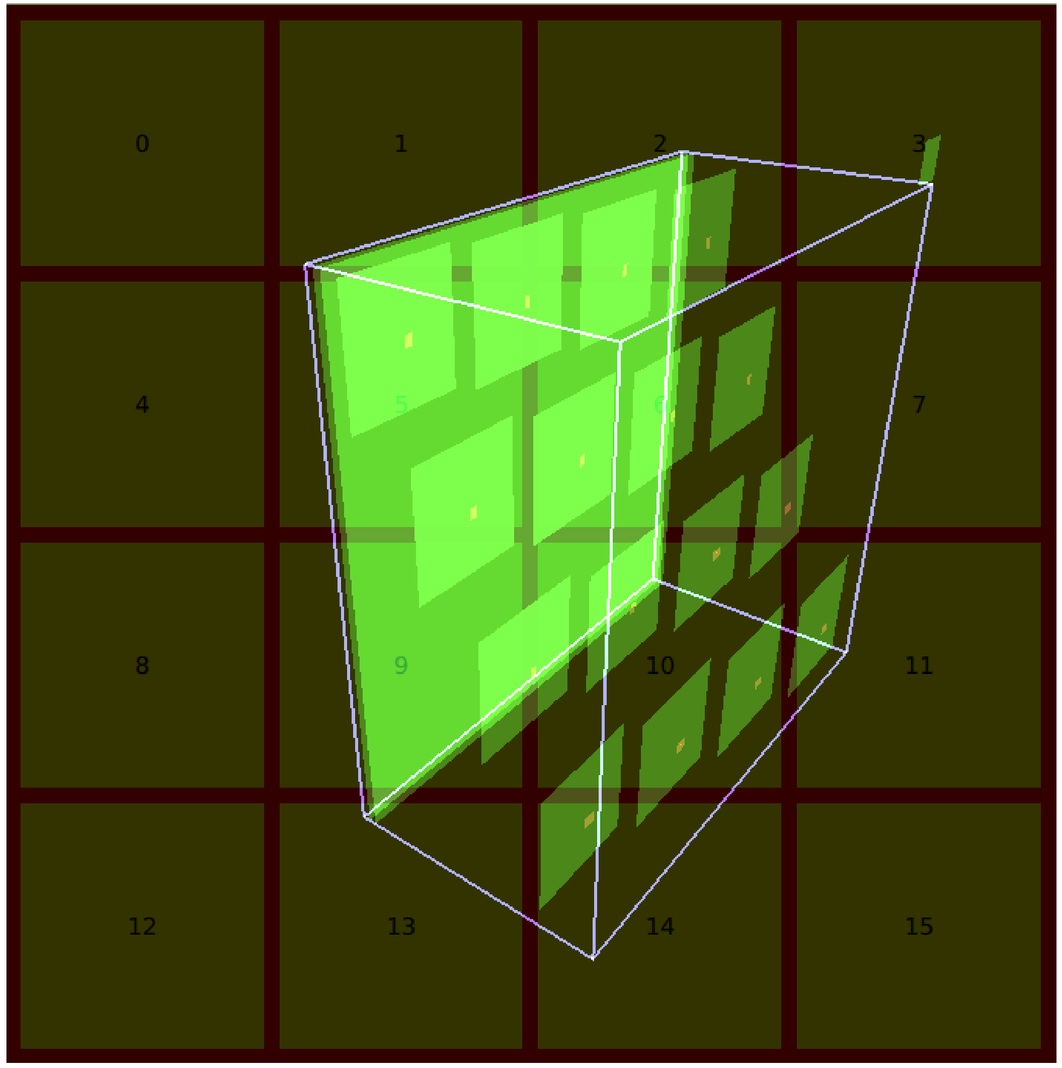
Rendering - Intro

- Rendering itself is rarely the culprit!
  - High CPU/GPU usage from other processes or threads
  - ListView scrollling instantiates new delegates
  - Timers in C++ or JS, event handling in C++
  - Use a CPU profiler and the QML profiler first to verify!

Rendering - Analyzing Frame Time

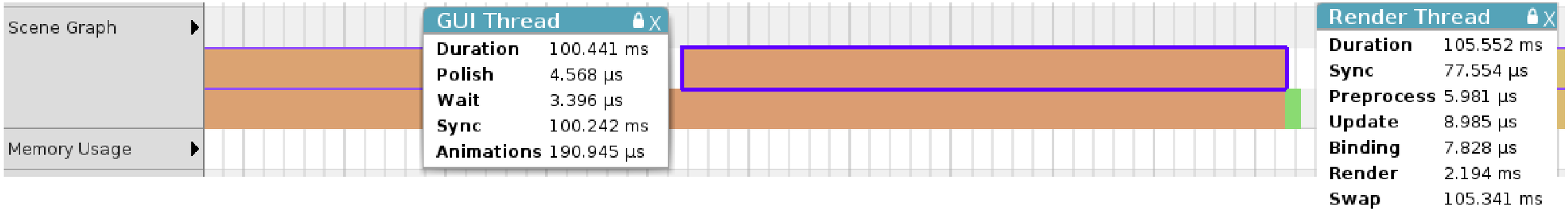
- See http://qt-project.org/doc/qt-5/qtquick-visualcanvas-scenegraph-renderer.html#performance for general tips to improve render performance
- Useful visualizations with *QSG_VISUALIZE*
  - batches
  - clip
  - overdraw
  - changes

Rendering - Visualizations



- *QSG_VISUALIZE=overdraw*
- No viewport clipping and occlusion culling in renderer!
- Make sure *visible* is false

Rendering - Measuring Frame Time




```
Render Thread: window=0x7fffa3aaf4b0, framedelta=51, sync=0, first render=104, after final swap=1
 - Gui Thread: window=0x7fffa3aaf4b0, polish=0, lock=0, block/sync=100 -- animations=0
 - Breakdown of render time: preprocess=0, updates=0, binding=0, render=53, total=53
```

- QtCreator Enterprise or *QSG_RENDER_TIMING=1*
- *QSG_RENDER_LOOP=threaded*
- Measures **CPU** time
- No animations running -> 0 FPS

Rendering - Measuring Frame Time

- GUI Thread
  - **polish**: *QQuickItem::updatePolish()*
    - anchor and text layouting, canvas drawing, ...
  - **animations**: Advancing all animations (binding updates!)
  - **lock**: Posting sync request to render thread
  - **block/sync**: Wait for render thread to call *QQuickItem::updatePaintNode()*
    - Main/GUI thread will block while render thread busy!

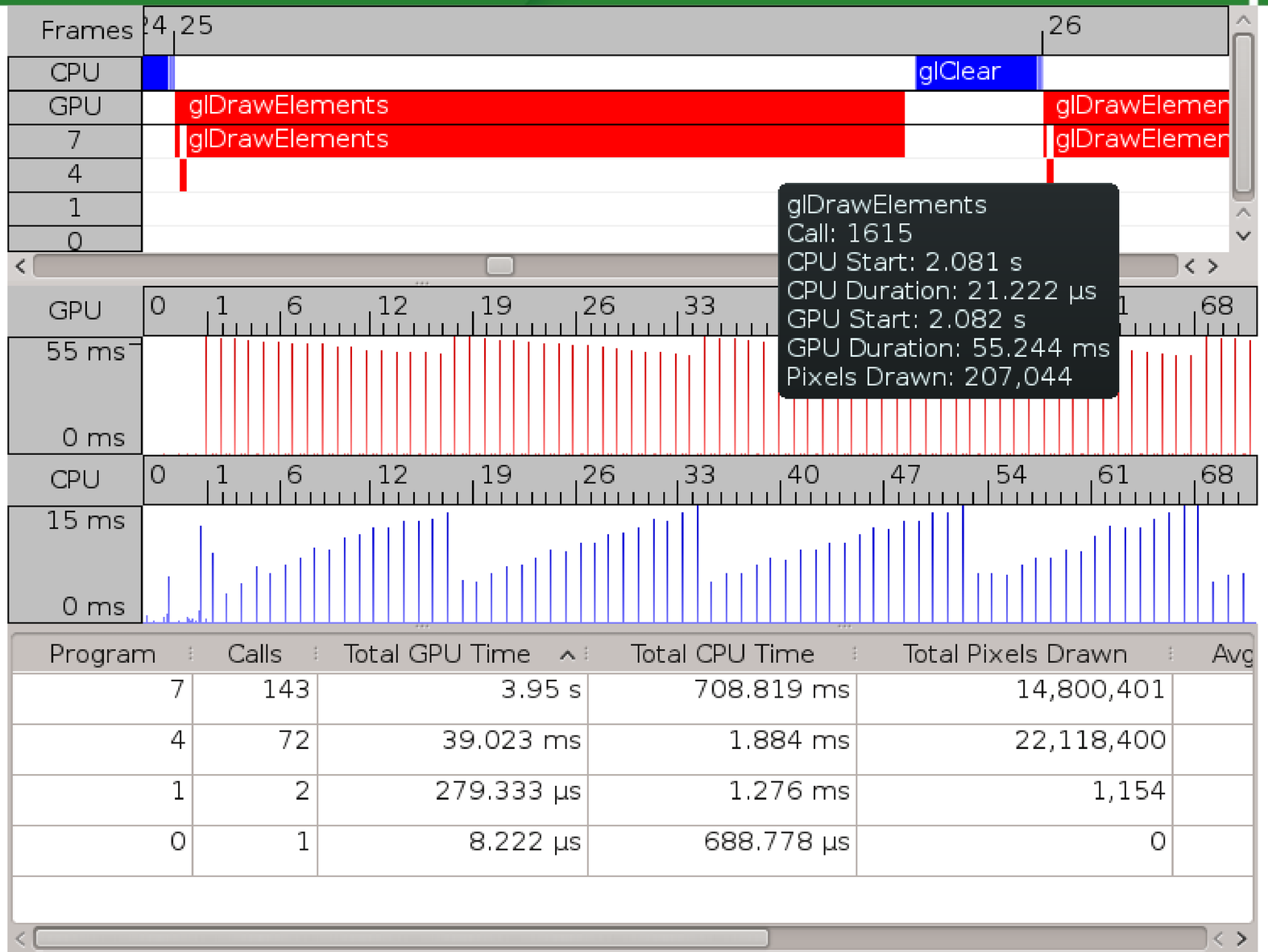
```
Render Thread: window=0x7fffa3aaf4b0, framedelta=51, sync=0, first render=104, after final swap=1
 - Gui Thread: window=0x7fffa3aaf4b0, polish=0, lock=0, block/sync=100 -- animations=0
 - Breakdown of render time: preprocess=0, updates=0, binding=0, render=53, total=53
```

Rendering - Measuring Frame Time

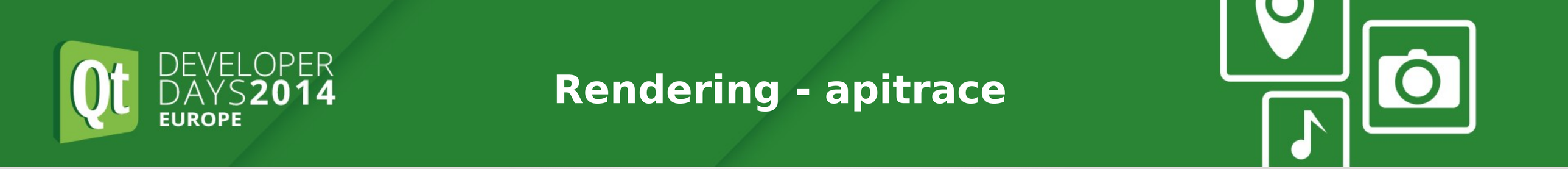
- Render Thread
  - **framedelta**: 1000 / FPS
  - **sync**: Actual *QQuickItem::updatePaintNode()* call
  - **first render**: *CPU* render time
  - **final swap**: Swap time
- Caveat: swap time + render time >= 16ms with 60 Hz vsync
- Caveat: Some drivers wait in first GL call of next frame, not in *glSwapBuffers()*!

```
Render Thread: window=0x7fffa3aaf4b0, framedelta=51, sync=0, first render=104, after final swap=1
 - Gui Thread: window=0x7fffa3aaf4b0, polish=0, lock=0, block/sync=100 -- animations=0
 - Breakdown of render time: preprocess=0, updates=0, binding=0, render=53, total=53
```

Rendering - apitrace



Program	Calls	Total GPU Time	Total CPU Time	Total Pixels Drawn	Avg
7	143	3.95 s	708.819 ms	14,800,401	
4	72	39.023 ms	1.884 ms	22,118,400	
1	2	279.333 µs	1.276 ms	1,154	
0	1	8.222 µs	688.778 µs	0	



File Edit View Trace

Events

```
glDisableVertexAttribArray(2)
glUseProgram(7)
glUniformMatrix4fv(0, 1, GL_FALSE, [0.003125, 0, 0, 0, 0, -0.004166
glActiveTexture(GL_TEXTURE0)
glBindTexture(GL_TEXTURE_2D, 1)
glActiveTexture(GL_TEXTURE0)
glDisable(GL_CULL_FACE)
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 16, NULL)
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 16, 0x8)
glDrawElements(GL_TRIANGLE_STRIP, 6, GL_UNSIGNED_SHORT, 0x
glDisableVertexAttribArray(0)
glDisableVertexAttribArray(1)
glDisable(GL_CULL_FACE)
glDisable(GL_STENCIL_TEST)
glDisable(GL_SCISSOR_TEST)
glBindBuffer(GL_ARRAY_BUFFER, 0)
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0)
glDisable(GL_SCISSOR_TEST)
glXSwapBuffers(0xa2a7c0, 109051915)
Frame 18  (53 calls)
Frame 19  (53 calls)
```

Current State

Parameters | Shaders | Surfaces | Uniforms

Select a shader: GL_FRAGMENT_SHADER

```
1  #define lowp
2  #define mediump
3  #define highp
4
5                          varying highp vec2 coord;
6                          uniform sampler2D src;
7                          void main() {
8                              int sum = 0;
9                              for(int i=0;i<20;++i)
10                               for(int j=0;j<20;++j)
11                                   for(int k=0;k<8;++k)
12                                       sum += i * j * k;
13
14                             lowp vec4 tex = texture2D(src, coord);
15                             gl_FragColor = vec4(tex.rgb, float(sum));
16                         }
```

Details View. Frame 17, Call 1239

1239) glDrawElements(mode = GL_TRIANGLE_STRIP, count = 6, type = GL_UNSIGNED_SHORT, indices = 0x40)

Details View. Frame 17, Call 1239 | Errors

Rendering - apitrace

- Traces and times OpenGL calls on CPU and GPU
- Shows complete GL state, including buffers and shaders
- Useful when integrating custom items into QtQuick
- Useful when working on the scenegraph renderer itself
- *Usage:*
  - *apitrace trace* to record
  - *qapitrace* to visualize and play back

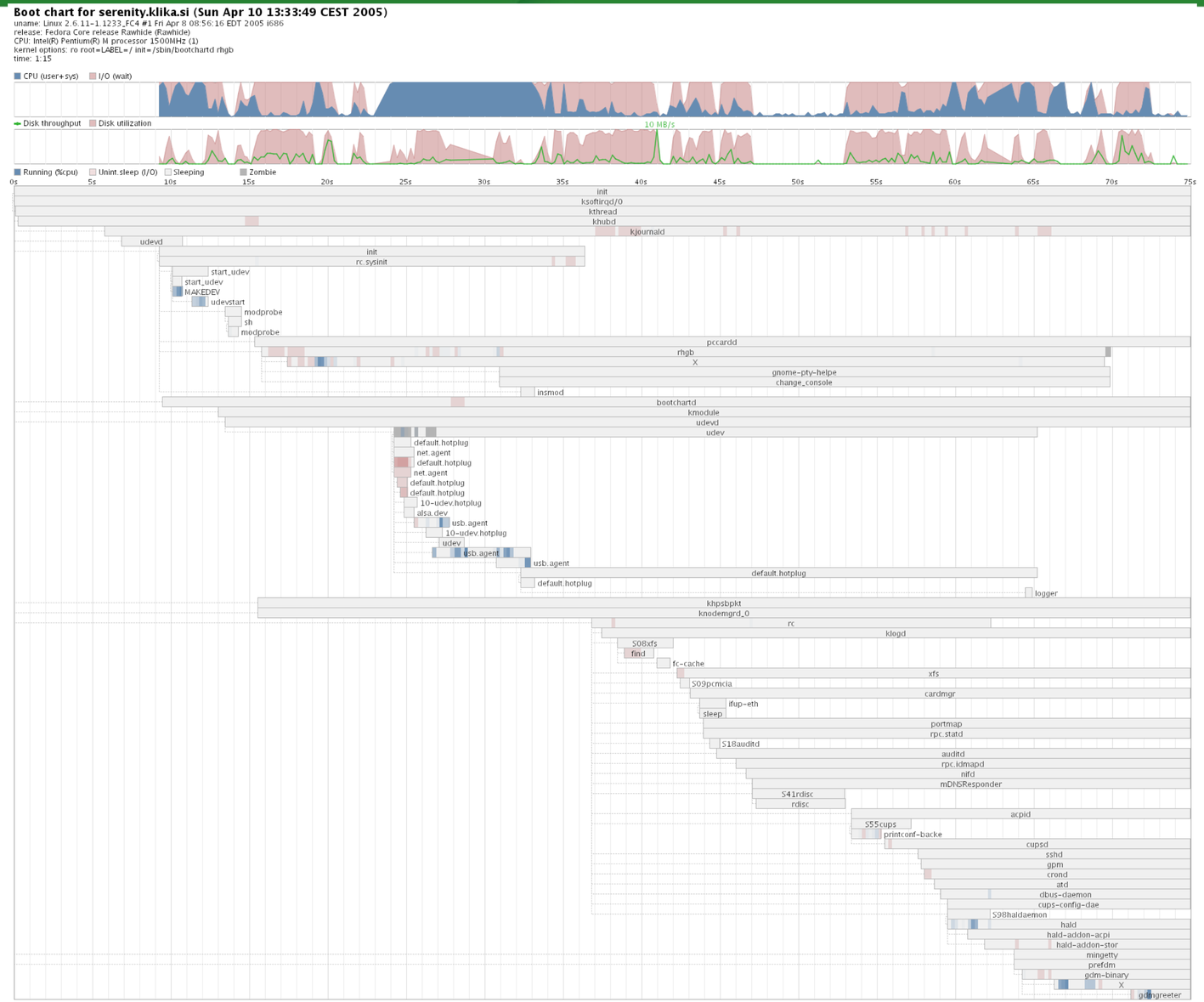
Responsiveness

Responsiveness

- Usually starts in QtQuick signal handlers like *onClicked* or *onPressed*
- Mix of JS code, property/binding updates and calls into C++
- Measure only relevant time period
- Start with QML Profiler, descent into CPU profiler if needed
- May load new view
  - Similar analysis as startup time
  - Loader: startup time vs reaction time

Boot Duration

Boot Duration - bootchart



Power Usage

Power Usage - powertop

```
    PowerTOP version 1.0       (C) 2007 Intel Corporation

Cn          Avg residency (5s)  Long term residency avg
C0 (cpu running)        ( 3.8%)
C1                0.0ms ( 0.0%)                   0.0ms
C2                4.4ms (57.3%)                   4.4ms
C3               10.0ms (31.1%)                  10.0ms
C4                2.3ms ( 7.7%)                   2.3ms

Wakeups per second :  193.6
Power usage (ACPI estimate) :  13.0 W (6.5 hours left)

Top causes for wakeups:
  35.2%        <interrupt> : i8042
  28.4%        <interrupt> : yenta, i915@pci:0000:00:02.0
  13.6%        <interrupt> : ipw2200, Intel 82801DB-ICH4
   4.6%               Xorg : do_setitimer (it_real_fn)
   3.7%        firefox-bin : schedule_timeout (process_timeout)
   3.5%              xchat : schedule_timeout (process_timeout)
   1.6%        firefox-bin : schedule_timeout (process_timeout)
   1.3%     gnome-terminal : schedule_timeout (process_timeout)
   1.1%    gnome-power-man : schedule_timeout (process_timeout)
   1.1%            emerald : schedule_timeout (process_timeout)

Suggestion: Enable the CONFIG_USB_SUSPEND kernel configuration option.
This option will automatically disable UHCI USB when not in use, and may
save approximately 1 Watt of power.
```

Power Usage - Others

- **powertop** to check for process wakeups and HW power usage
- **QML profiler** to check for unnecessary animations
- **Gammaray** timer top to check for unnecessary timers

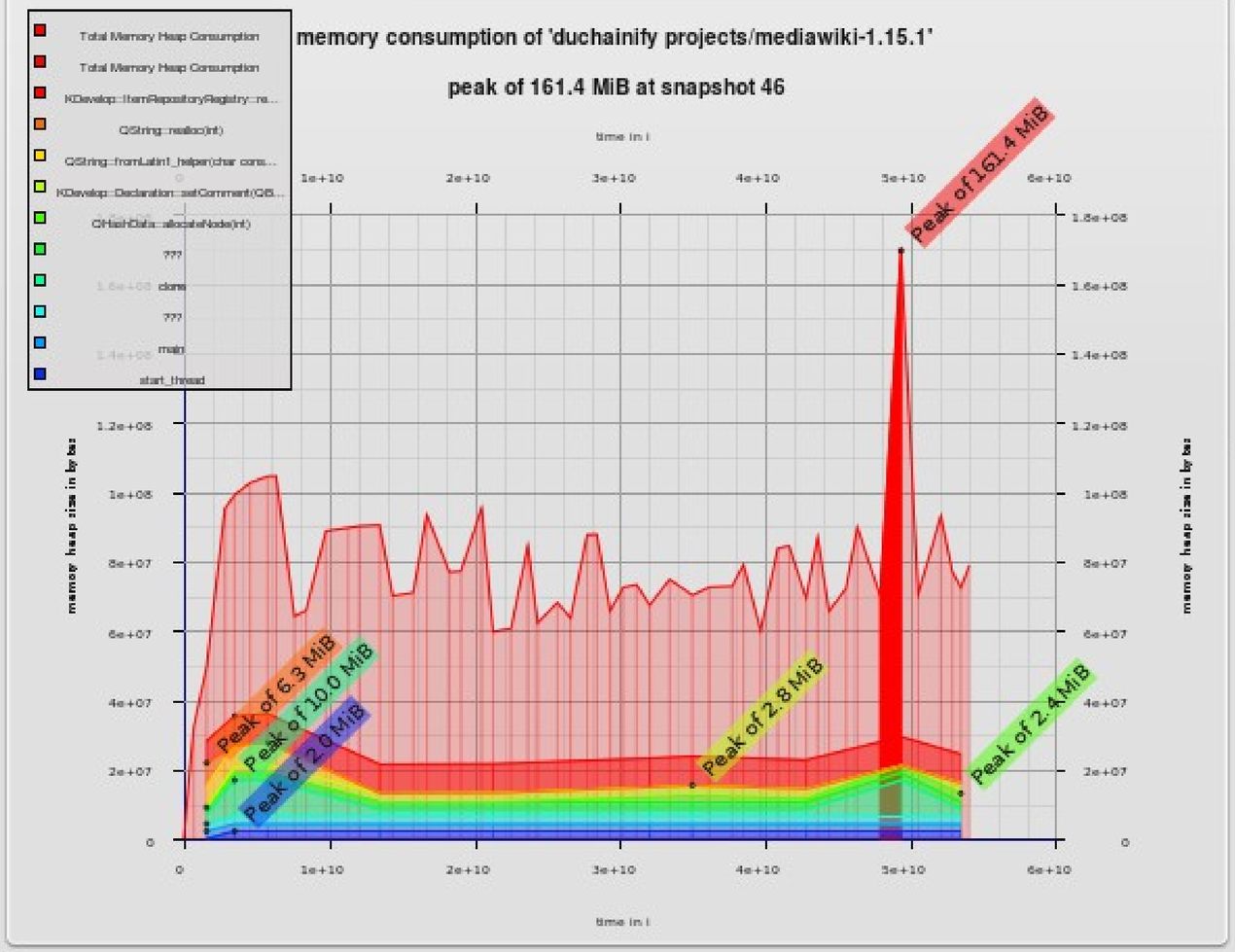
Memory Usage

Memory Usage - massif

File View Settings Help

Open | Close | toggle total cost graph | toggle detailed cost graph

Evolution of Memory Consumption | Detailed Snapshot Analysis



Massif Data

filter

Cost	Location
64.8 MiB	snapshot #24
60.6 MiB	snapshot #25
83.4 MiB	snapshot #26
83.5 MiB	snapshot #27
62.5 MiB	snapshot #28
69.0 MiB	snapshot #29
69.7 MiB	snapshot #30
+ 64.1 MiB	snapshot #31
71.1 MiB	snapshot #32
+ 66.8 MiB	snapshot #33
69.0 MiB	snapshot #34
69.3 MiB	snapshot #35
75.2 MiB	snapshot #36
57.1 MiB	snapshot #37
79.7 MiB	snapshot #38
80.4 MiB	snapshot #39
+ 66.0 MiB	snapshot #40
82.7 MiB	snapshot #41
62.4 MiB	snapshot #42
68.7 MiB	snapshot #43
85.5 MiB	snapshot #44
66.6 MiB	snapshot #45
− 161.4 MiB	snapshot #46 (peak)
+ 85.3 MiB	KDevPG::BlockType::init(int) (kdev-pg-...
19.4 MiB	in 1654 places, all below massif's thres...
+ 10.0 MiB	KDevPG::TokenStreamBase<Php::Toke...
+ 8.0 MiB	KDevelop::ItemRepository<Utils::SetN...
+ 2.8 MiB	KDevelop::Bucket<Repositories::String...
+ 2.0 MiB	KDevelop::ItemRepository<KDevelop::...

Memory Usage - Others

- **massif** to track C++ heap allocations
- **QML Profiler** (enterprise) to track JS memory usage
- QML engine: ?

Thank you!

Questions?

Thomas McGuire - KDAB - thomas@kdab.com