Tracy:
A Profiler You Don't Want to Miss

MARCOS SLOMP

Tracy: a profiler you don't want to miss

Marcos Slomp

Senior Research Engineer II

Why Tracy?

Real-time workflow
(analyze application **while it is running**)

Precise measurements
(**nanosecond** resolution)

Negligible overhead
(a few **nanoseconds** per zone)

Cross-platform
(Windows, Linux, macOS, iOS, Android, WASM*)

Hybrid profiling capabilities
(**sampling** and/or **instrumentation**)
(**CPU** and **GPU** instrumentation)

Tracing capabilities
(values, messages, plots, allocations, ...)

Hassle-free integration
(single source* file, and header*)

Free and **Open source**

VISA MasterCard AMERICAN EXPRESS DISCOVER NETWORK

PAY WHAT YOU WANT

$0.00

Exc. Internet fees
Inc. source code

1-555-PROFILE

FREE Trial! Instant Delivery!

Lifetime Money Back Guarantee (less S&H).

https://github.com/wolfpld/tracy

Why Tracy?

Real-time workflow
(analyze application **while it is running**)

Precise measurements
(**nanosecond** resolution)

Negligible overhead
(a few **nanoseconds** per zone)

Cross-platform
(Windows, Linux, macOS, iOS, Android, WASM*)

Hybrid profiling capabilities
(**sampling** and/or **instrumentation**)

Tracing capabilities
(values, messages, plots, allocations, ...)

Free and **Open source**

Why Tracy?

Real-time workflow
(analyze application **while it is running**)

Precise measurements
(**nanosecond** resolution)

Negligible overhead
(a few **nanoseconds** per zone)

Cross-platform
(Windows, Linux, macOS, iOS, Android, WASM*)

Hybrid profiling capabilities
(**sampling** and/or **instrumentation**)

Tracing capabilities
(values, messages, plots, allocations, ...)

Free and **Open source**

DISCLAIMER: this talk is ***NOT*** about “Tracy vs. other tools”

Telemetry	http://www.radgametools.com/telemetry.htm
Microprofile	https://github.com/jonasmr/microprofile
Microprofile 2	https://github.com/zeux/microprofile
Optick (Brofiler)	https://github.com/bombomby/optick
Palanteer	https://github.com/dfeneyrou/palanteer
Orbit Profiler	https://github.com/google/orbit
SuperLuminal	https://superluminal.eu
Perfetto	https://perfetto.dev
MicroProfiler	https://github.com/tyoma/micro-profiler
EasyProfiler	https://github.com/yse/easy_profiler
Oprofile	https://oprofile.sourceforge.io/examples/
Coz	https://github.com/plasma-umass/coz
VerySleepy	https://github.com/VerySleepy/verysleepy
LukeStackwalker	http://lukestackwalker.sourceforge.net
dprofiler	https://github.com/xwlan/dprofiler
hotspot	https://github.com/KDAB/hotspot
heaptrack	https://github.com/KDE/heaptrack
spall	https://handmade.network/p/333/spall/
geiger	https://github.com/david-grs/geiger
Intel IACA	https://www.intel.com/content/www/us/en/developer/articles/tool/architecture-code-analyzer.html

The Obligatory Quote

premature optimization is the root of all evil.

Donald E. Knuth

The Obligatory Quote, in proper context

"There is no doubt that the grail of efficiency leads to abuse. Programmers **waste enormous amounts of time** thinking about, or **worrying** about, the **speed of noncritical parts** of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We *should* **forget about small efficiencies**, say about 97% of the time: **premature optimization is the root of all evil.** Yet we **should not pass up our opportunities in that critical 3%**."

Donald E. Knuth. 1974. Structured Programming with go to Statements. ACM Computing Surveys 6, 4 (Dec. 1974), 261–301.

Mathieu Ropert

https://www.youtube.com/watch?v=dTOaepIXW4s

"The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times"

"

10

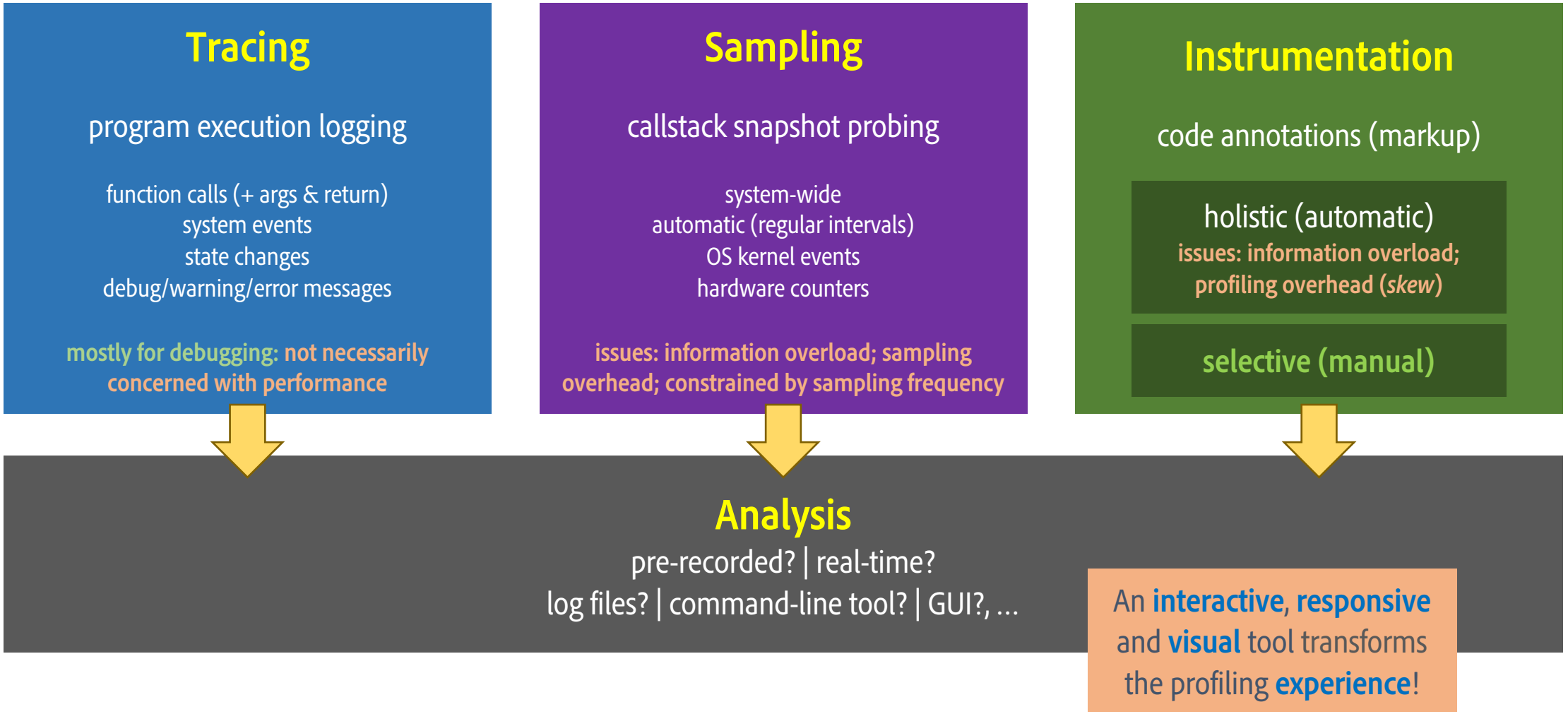
"C++ programmers talk all the time about **efficiency**, but they generally don't know how to **measure**."

- Bjarne Stroustrup

CppCon 2023 Keynote

(literally three days ago!)

Closing the Profiling Zoo



Closing the Profiling Zoo

Tracing

program execution loggi

function calls (+ args & return
system events
state changes
debug/warning/error message

mostly for debugging: not neces
concerned with performanc

Instrumentation

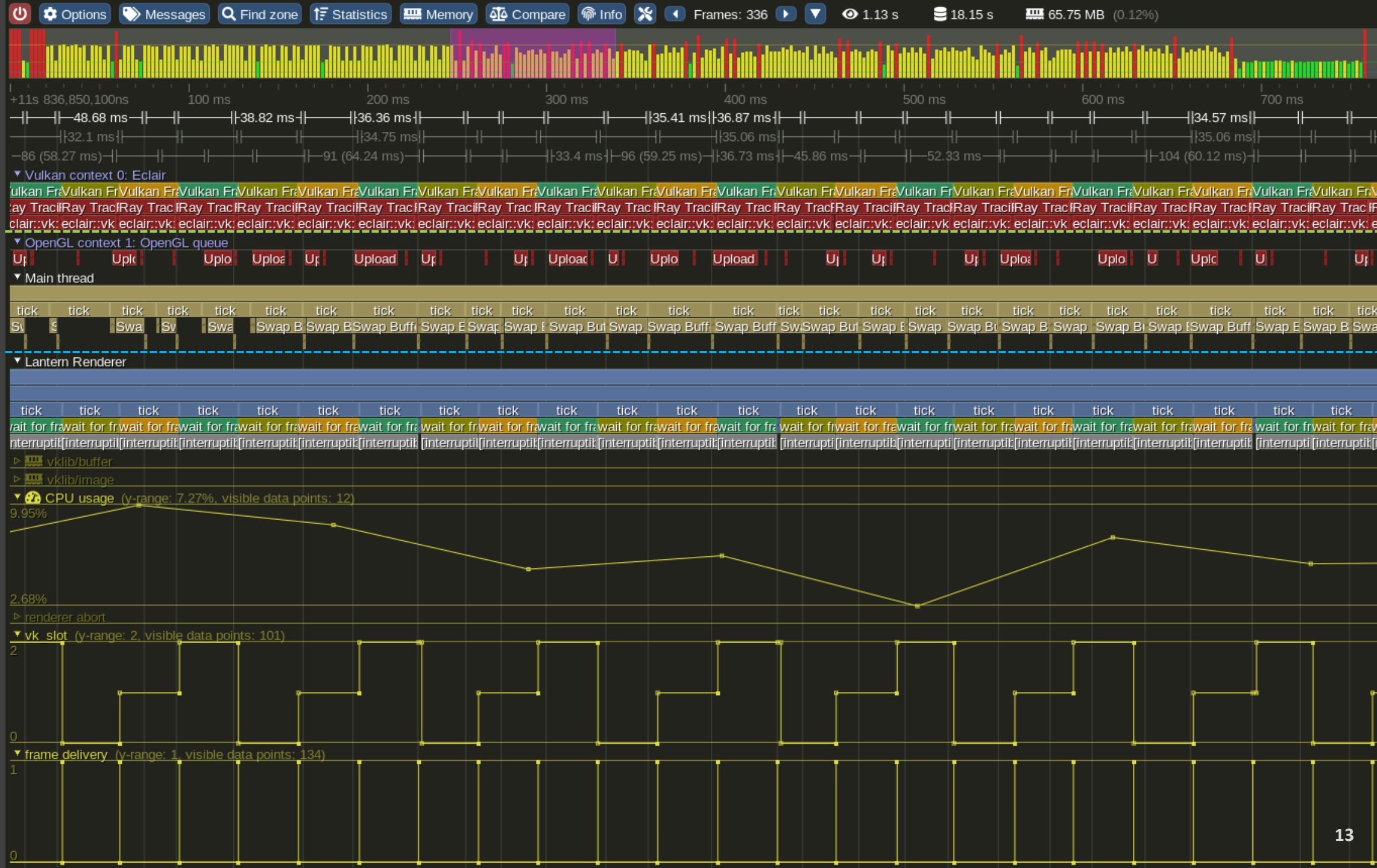
code annotations

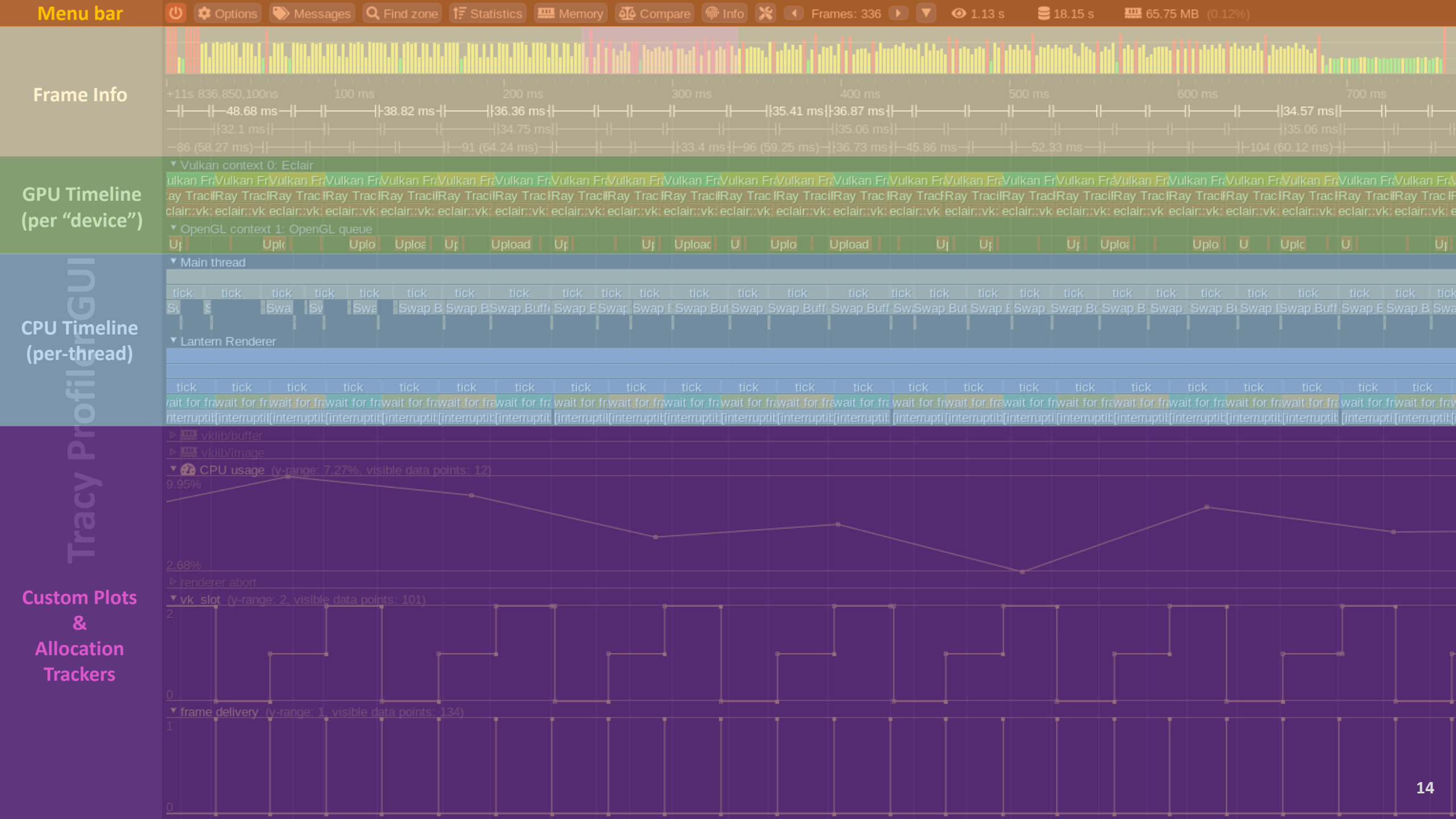
holistic (automatic)
ssues: information overload;
profiling overhead (*skew*)

selective (manual)

teractive, responsive
isual tool transforms
rofiling experience!

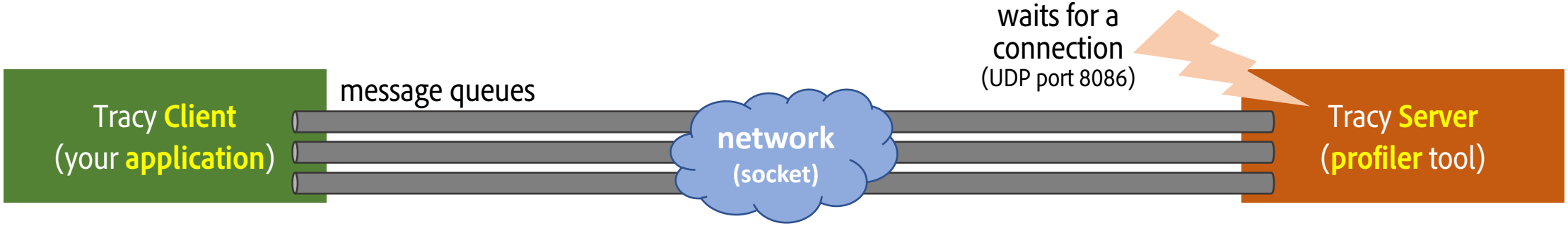
Tracy Profiler GUI




Menu bar
Options
Messages
Find zone
Statistics
Memory
Compare
Info
Frames: 336
1.13 s
18.15 s
65.75 MB (0.12%)
Frame Info
+11s 836,850,100ns
100 ms
200 ms
300 ms
400 ms
500 ms
600 ms
700 ms
48.68 ms
38.82 ms
36.36 ms
35.41 ms
36.87 ms
34.57 ms
32.1 ms
34.75 ms
35.06 ms
35.06 ms
86 (58.27 ms)
91 (64.24 ms)
33.4 ms
96 (59.25 ms)
36.73 ms
45.86 ms
52.33 ms
104 (60.12 ms)
GPU Timeline (per “device”)
Vulkan context 0: Eclair
OpenGL context 1: OpenGL queue
CPU Timeline (per-thread)
Main thread
Lantern Renderer
tick
Swap
wait for fr
interruptil
Tracy Profiler GUI
Custom Plots & Allocation Trackers
vklib/buffer
vklib/image
CPU usage (y-range: 7.27%, visible data points: 12)
9.95%
2.68%
renderer abort
vk_slot (y-range: 2, visible data points: 101)
2
0
frame delivery (y-range: 1, visible data points: 134)
1
0


QUAKE II

Tracy Architecture



Basic integration (the *responsible* way):

1. simply `#include <Tracy.hpp>` whenever needed
2. compile `TracyClient.cpp` alongside your application
3. enable Tracy by defining `TRACY_ENABLE` in your build

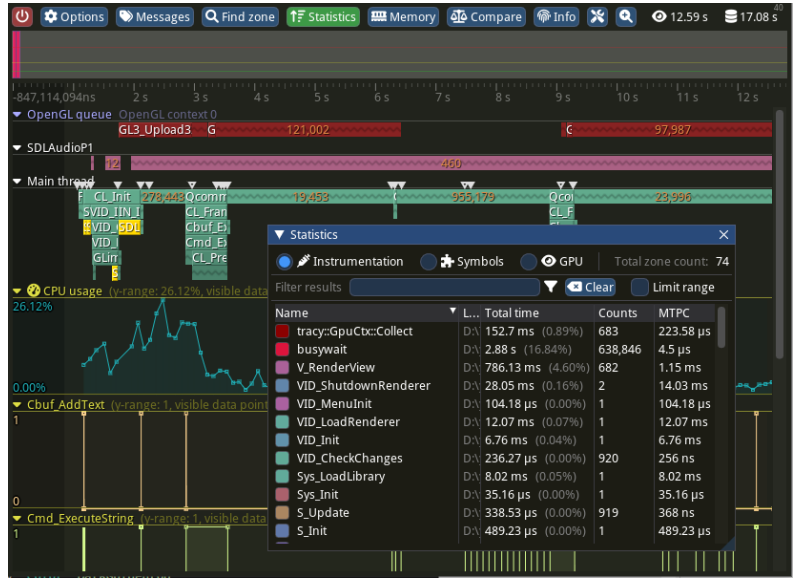
Quick integration (the *YOLO* way):

```
#define TRACY_ENABLE
#include <Tracy.hpp>
```

anywhere where needed

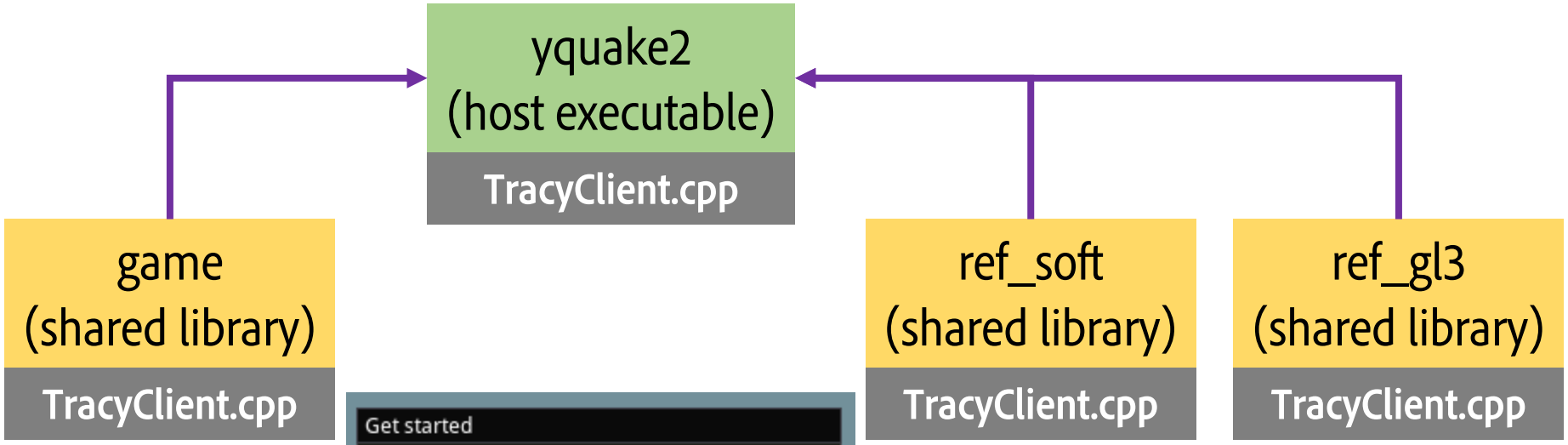
```
#define TRACY_ENABLE
#include <TracyClient.cpp>
```

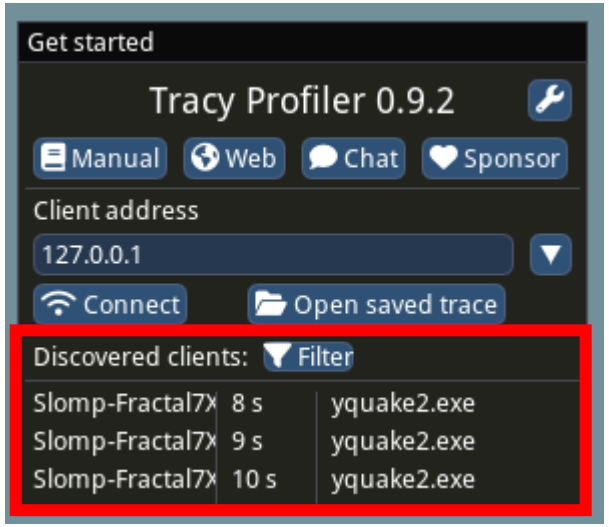
in *one* of the source files of your project



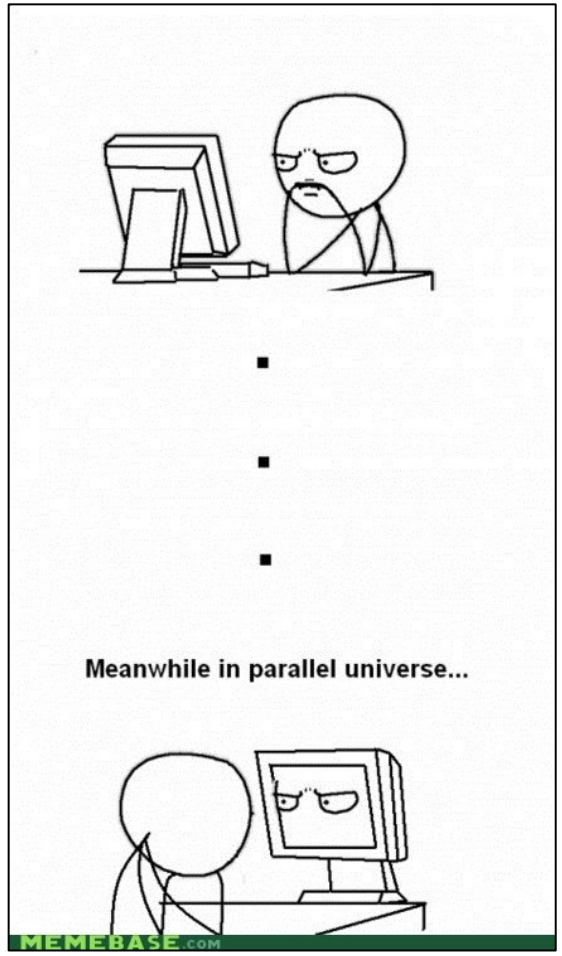
download pre-built binary (**Windows-only**):
https://github.com/wolfpld/tracy/releases
or **build it yourself**!

Tracy integration across shared libraries

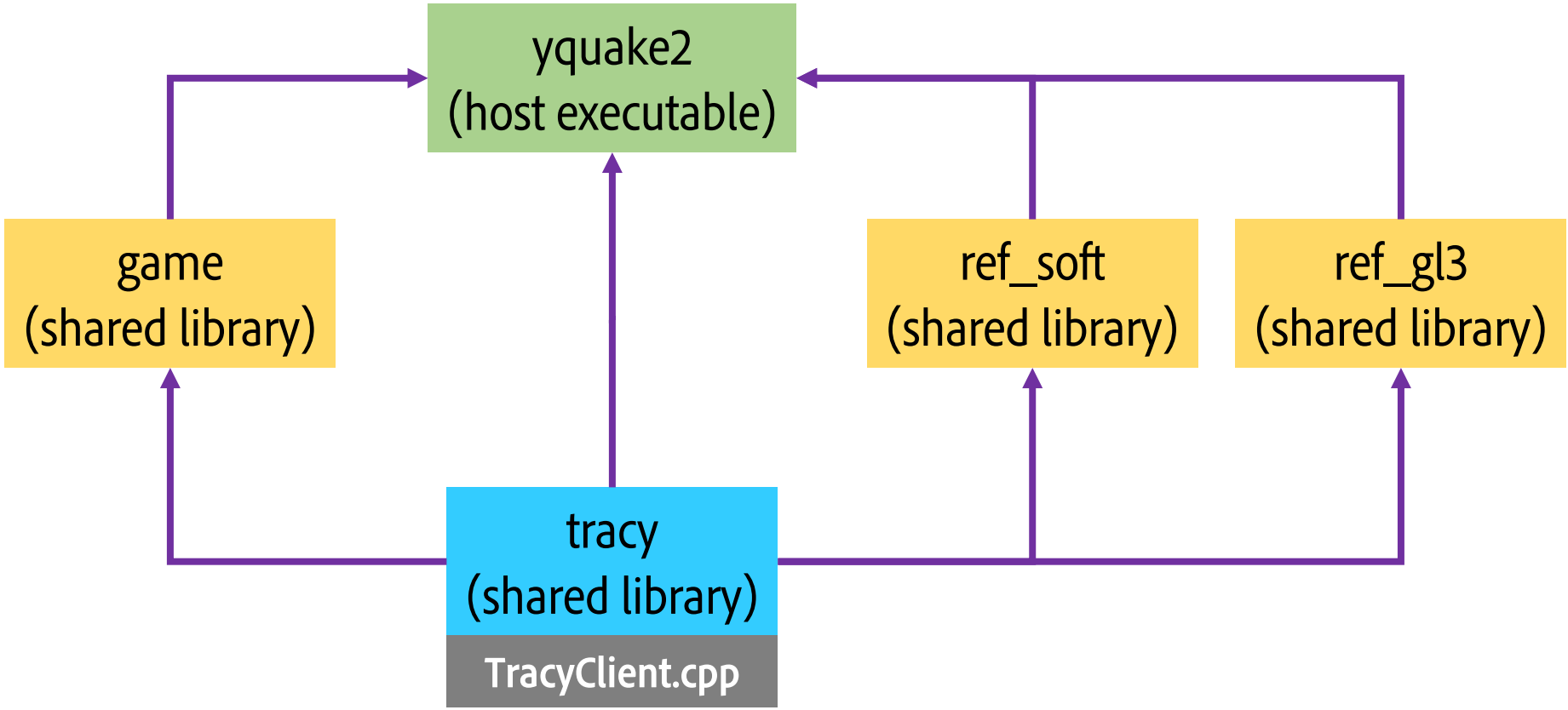




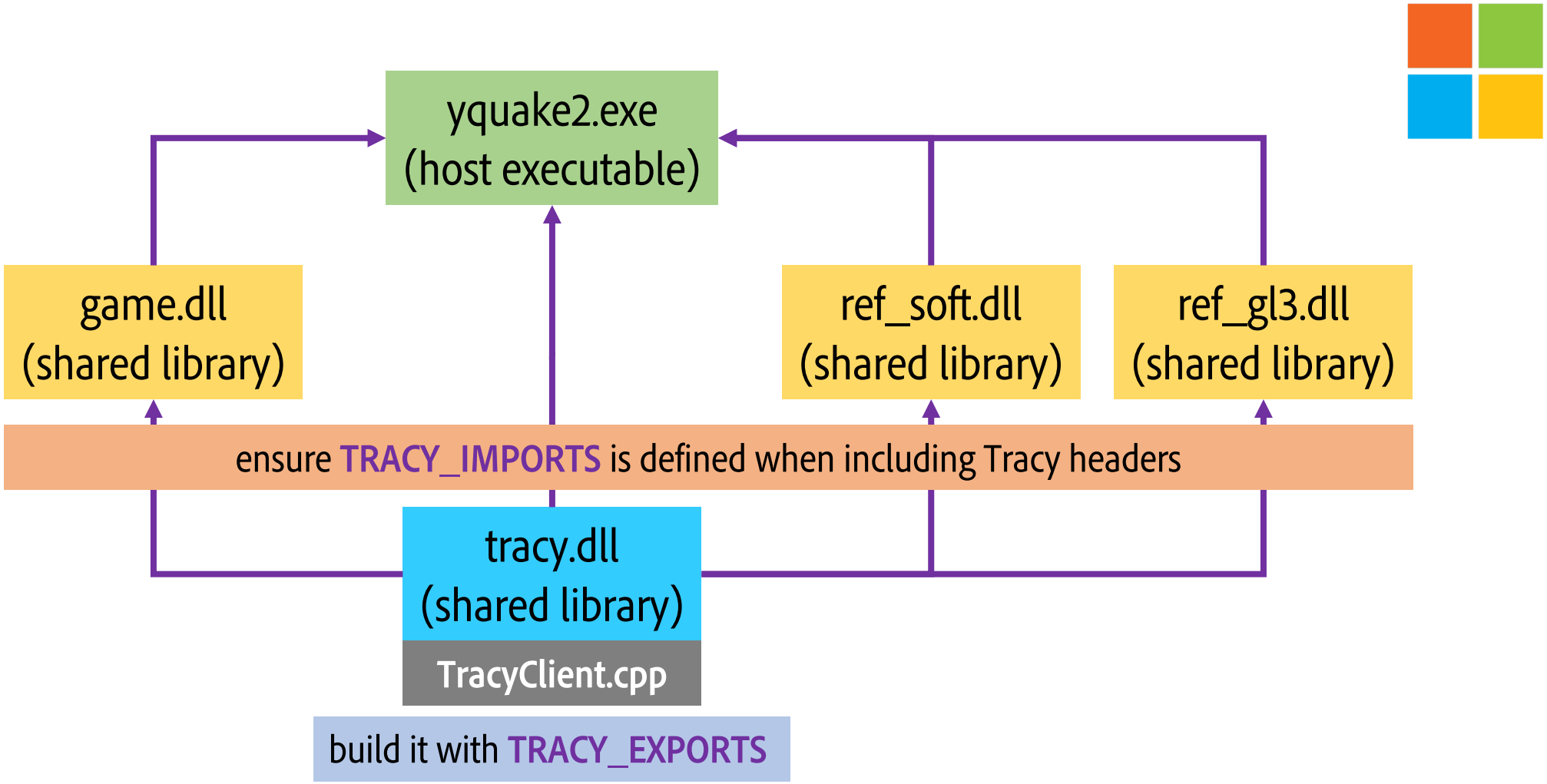
Behold the Tracy Multiverse!



Tracy integration across shared libraries



Tracy integration across shared libraries (Windows)



Building the Tracy Profiler from source

1. Obtain the dependencies (capstone, glfw, freetype):
<tracy-repo>\vcpkg\install_vcpkg_dependencies.bat

2. Build with the Visual Studio solution:
<tracy-repo>\profiler\build\win32\Tracy.sln

Tracy.exe

1. Obtain the dependencies (capstone, freetype) (glfw or libxkbcommon, wayland, libglvnd, and maybe libegl too):
(use whatever package manager your distro provides)

2. Build with make:
<tracy-repo>/profiler/build/unix $ make –j8

Tracy-release

1. Obtain the dependencies (capstone, glfw, freetype):
$ brew install capstone glfw freetype

2. Build with make:
<tracy-repo>/profiler/build/unix $ make –j8

Tracy-release

Zoning...

```
{ this is a zone scope (RAII) }

int foo()
{
    ...
    ...
    if (!spoon) {
        ...
        ...
    }
    ...
    {
        ...
        ...
    }
}
```

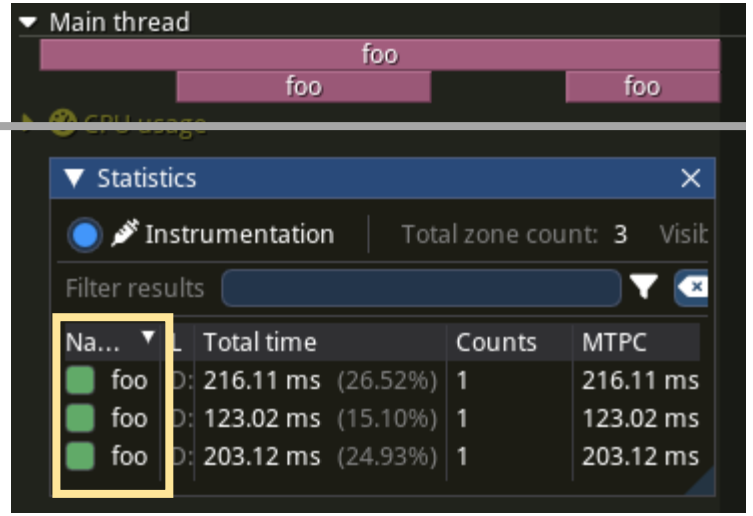
Source: https://www.mobygames.com/game/657/simcity-2000/promo/group-17786/image-185051/

Tracy Zoning 101

```
int foo()
{
    ZoneScoped;
    ...
    if (!spoon) {
        ZoneScoped;
        ...
    }
    ...
    {
        ZoneScoped;
        ...
    }
}
```

thread **timeline**

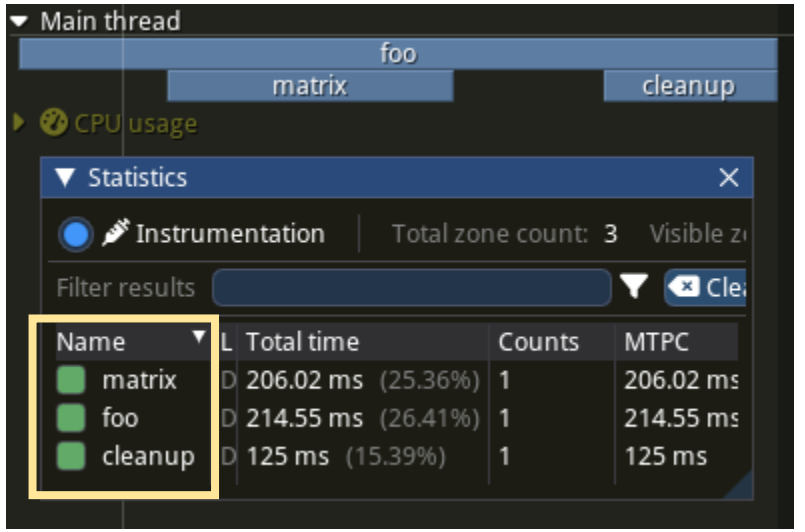
zone **statistics**



- **ZoneScoped**
  - the "**bread-and-butter**" of Tracy
  - only **1 zone per scope**
  - zone **name is inferred** (enclosing function name)
  - zone **color is automatic** (per-thread and depth)

Tracy Zoning 101

```
int foo()
{
    ZoneScoped;
    ...
    if (!spoon) {
        ZoneScopedN("matrix");
        ...
    }
    ...
    {
        ZoneScopedN("cleanup");
        ...
    }
}
```



- **ZoneScopedN**
  - only 1 zone per scope
  - zone color is automatic (per-thread and depth)
  - **you name it!** (must be ***persistent* string literal**)

Tracy manual, Section 3.1

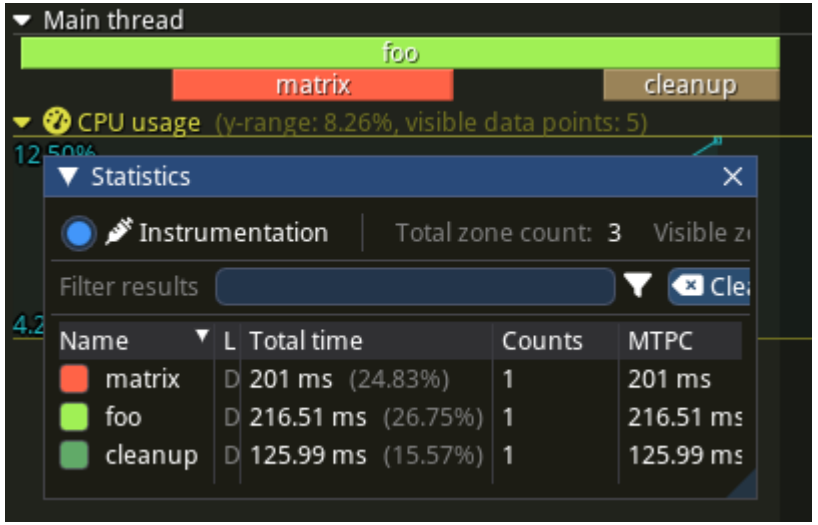
the provided string data must be accessible at any time in program execution (*this also includes the time after exiting the* `main` *function*). The string also cannot be changed. This basically means that the only option is to use a string literal

Tracy Zoning 101

```
int foo()
{
    ZoneScopedC(0xA0F055);
    ...
    if (!spoon) {
        ZoneScopedNC("matrix", Tomato);
        ...
    }
    ...
    {
        ZoneScopedN("cleanup");
        ...
    }
}
```

RRGGBB

tracy::Color::



- ZoneScoped[N]C
  - only 1 zone per scope
  - you [*may*] name it
  - **you pick the color**
    - hexadecimal RGB triplet: 0xRRGGBB
    - or X11 color names (trough **tracy::Color**)

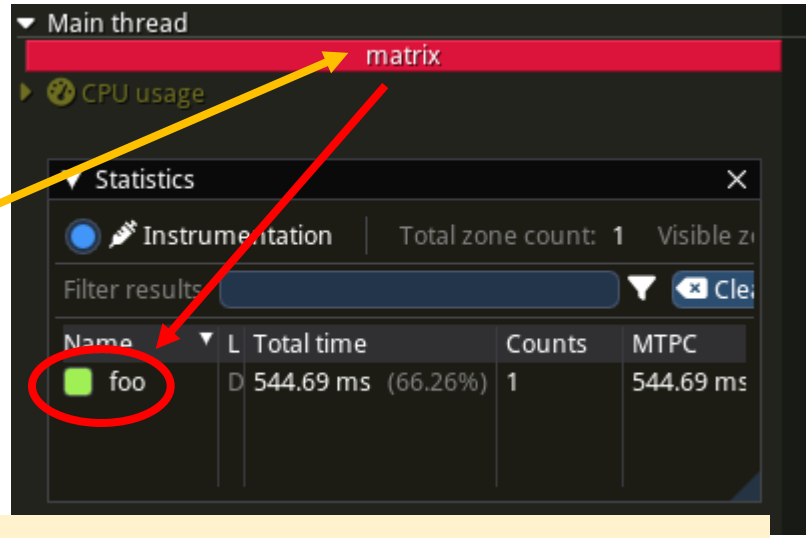

I KNOW TRACY
SHOW ME
ZoneScoped[N|C]

Tracy Zoning 201

```
int foo()
{
    ZoneScoped;
    ...
    if (!spoon) {
        ZoneName("matrix", 6);
        ...
    }
    ...
    if (error != 0) {
        ZoneColor(Crimson);
        ...
    }
}
```

strlen("matrix")

tracy::Color::

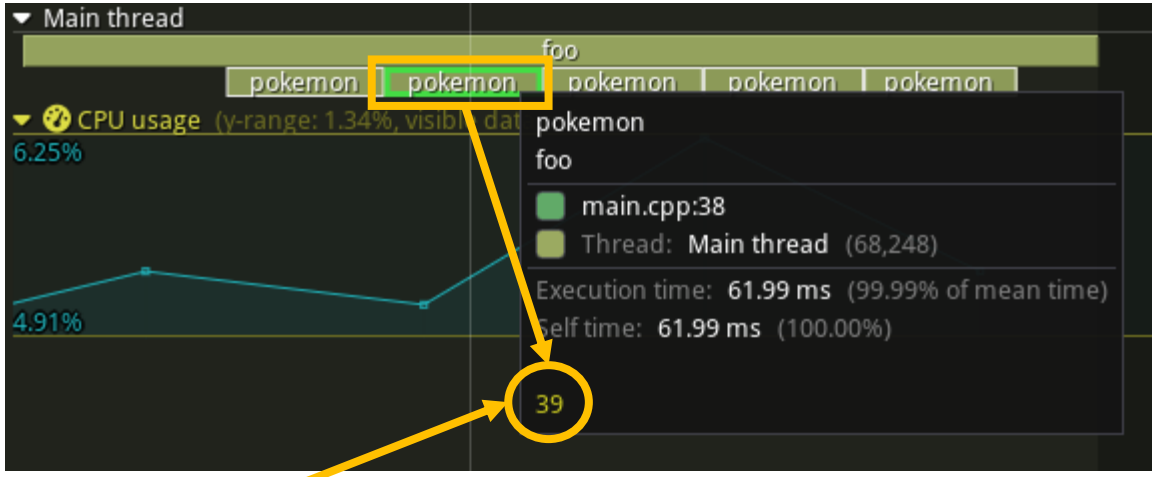


- ZoneName
  - programmatically change the zone name
  - can be non-persistent string (e.g., an std::string in the stack)

  Avoid! (it will confuse people unfamiliar with your code)
  - new name WON'T appear in profiler statistics!
- ZoneColor
  - programmatically change the zone color

Tracy Zoning 201

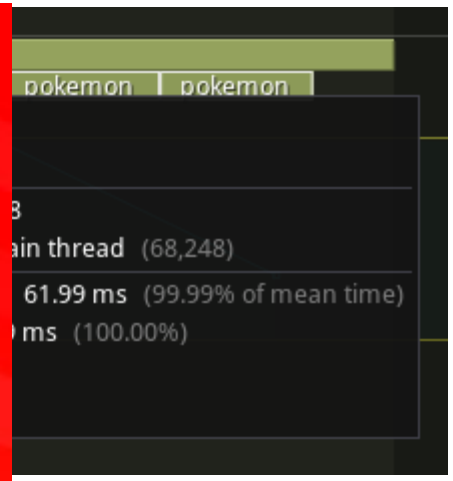
```
int foo()
{
    ZoneScoped;
    ...
    Pokedex pokedex;
    ...
    ZoneValue(pokedex.size());
    for (auto& pokemon : pokedex)
    {
        ZoneScopedN("pokemon");
        ZoneValue(pokemon.id);
        ...
    }
}
```



- ZoneValue
  - logs a value to the current zone scope
  - accepts ~~int64 or double~~ uint64

Tracy Zoning 201

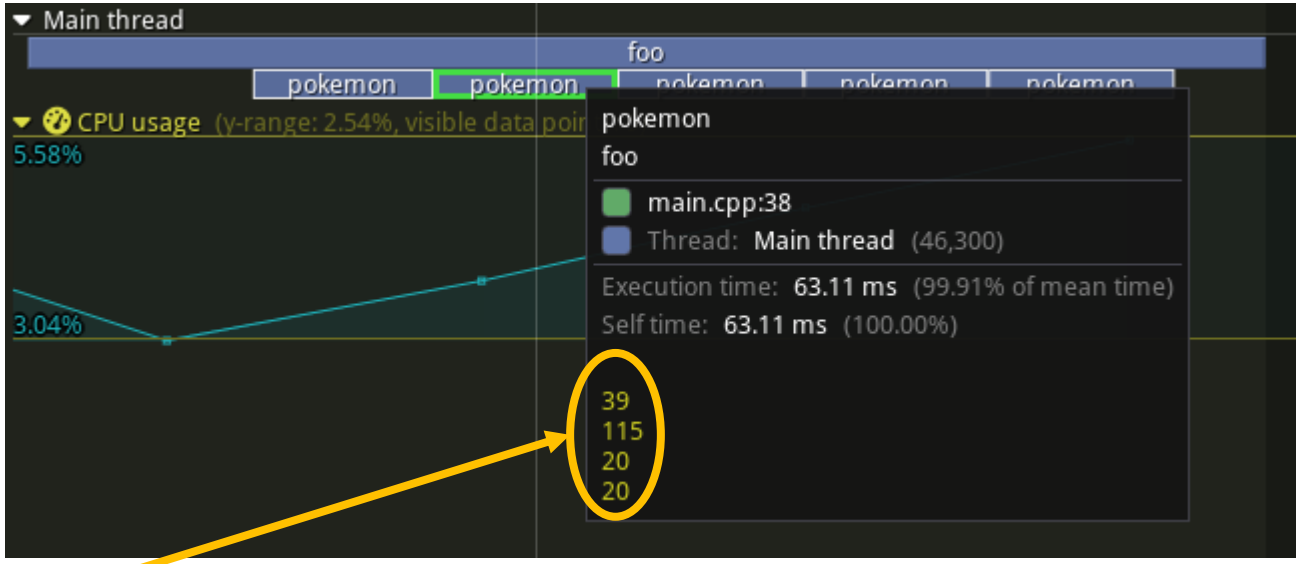
```
int foo()
{
    ZoneScoped;
    ...
    Pokedex pokedex;
    ...
    ZoneValue(pokedex
    for (auto& pokemo
    {
        ZoneScopedN("
        ZoneValue(pok
        ...
    }
}
```



ne scope

Tracy Zoning 201

```
int foo()
{
    ZoneScoped;
    Pokedex pokedex;
    for (auto& pokemon : pokedex)
    {
        ZoneScopedN("pokemon");
        ZoneValue(pokemon.id);
        ZoneValue(pokemon.hp);
        ZoneValue(pokemon.def);
        ZoneValue(pokemon.speed);
        ...
    }
}
```

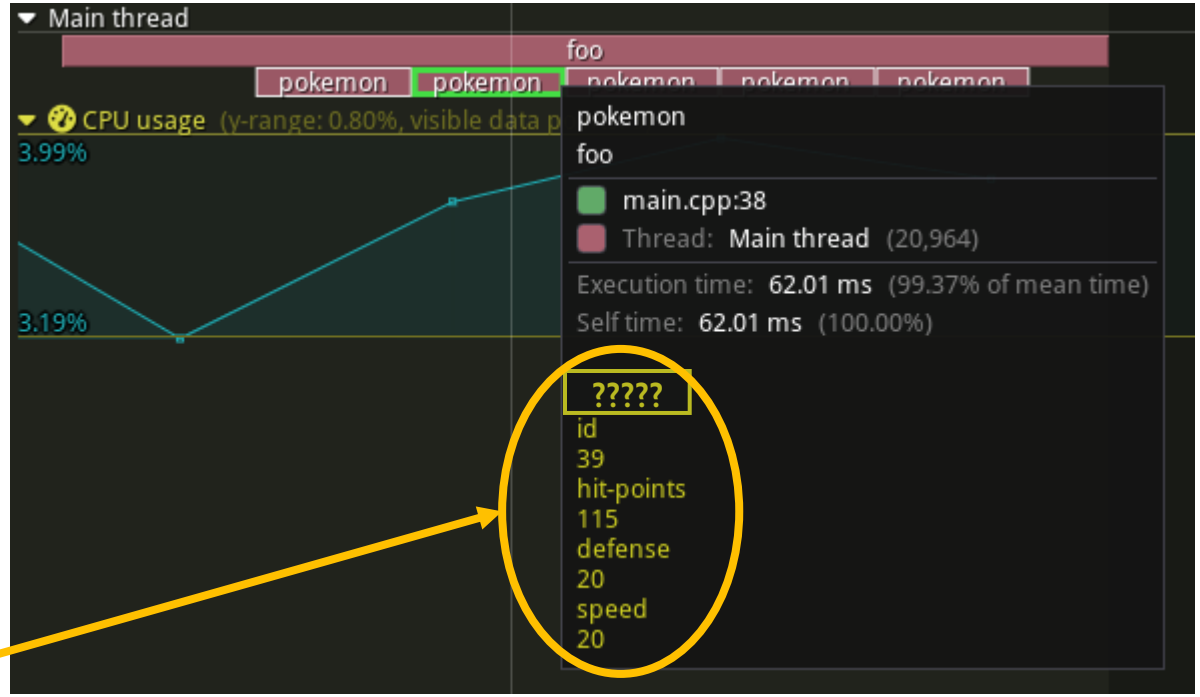


- **ZoneValue**
  - logs a value to the current zone scope
  - accepts ~~int64 or double~~ uint64
  - **multiple** values allowed!

Tracy Zoning 201

```
int foo()
{
    ZoneScoped;
    Pokedex pokedex;
    for (auto& pokemon : pokedex)
    {
        ZoneScopedN("pokemon");
        ...
        ZoneText("hit-points", 10);
        ZoneValue(pokemon.hp);
        ZoneText("defense", 7);
        ZoneValue(pokemon.def);
        ...
    }
}
```

strlen("hit-points")



- ZoneText
  - logs a string to the current zone scope
  - non-persistent string allowed
  - multiple texts allowed!

Tracy Zoning 201

```
int foo()
{
    ZoneScoped;
    Pokedex pokedex;
    for (auto& pokemon : pokedex)
    {
        ZoneScopedN("pokemon");
        ...
        ZoneText("hit-points", 10);
        ZoneValue(pokemon.hp);
        ZoneText("defense", 7);
        ZoneValue(pokemon.def);
        ...
    }
}
```

strlen("hit-points")

Source: https://www.reddit.com/r/pokemon/comments/3o5je7/jigglybuff/

- **ZoneText**
  - **logs a string** to the current zone scope
  - **non-persistent string** allowed
  - **multiple** texts allowed!

Tracy Zoning 301

```
int foo()
{
    ZoneScoped;
    ...
    ZoneNamedN(initzone, “init”, true);
    ...
    ZoneNamedN(setupzone, “setup”, true);
    ...
    ZoneNamedN(simzone, “simulate”, true);
    ...
    ZoneNamedN(logzone, “log”, false);
    ...
    ZoneNamedN(cleanzone, “cleanup”, true);
    ...
}
```

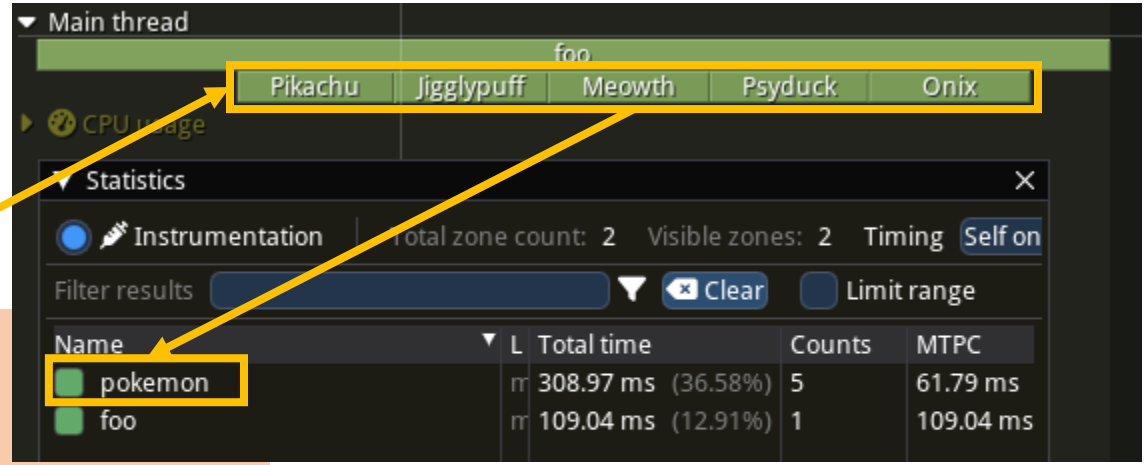
- ZoneNamed[N|C]
  - have multiple zones in the same scope
  - each “zone identifier” must be unique in that scope
  - creates a zone stack, not a partition
  - offers “active” argument to toggle a zone (on/off) programmatically

Tracy Zoning 301

ZoneScoped[N|C|S]	ZoneNamed[N|C|S]
ZoneColor(…)	ZoneColorV(varname, …)
ZoneValue(…)	ZoneValueV(varname, …)
ZoneText(…)	ZoneTextV(varname, …)
ZoneName(…)	ZoneNameV(varname, …)
ZoneIsActive()	ZoneIsActiveV(varname)

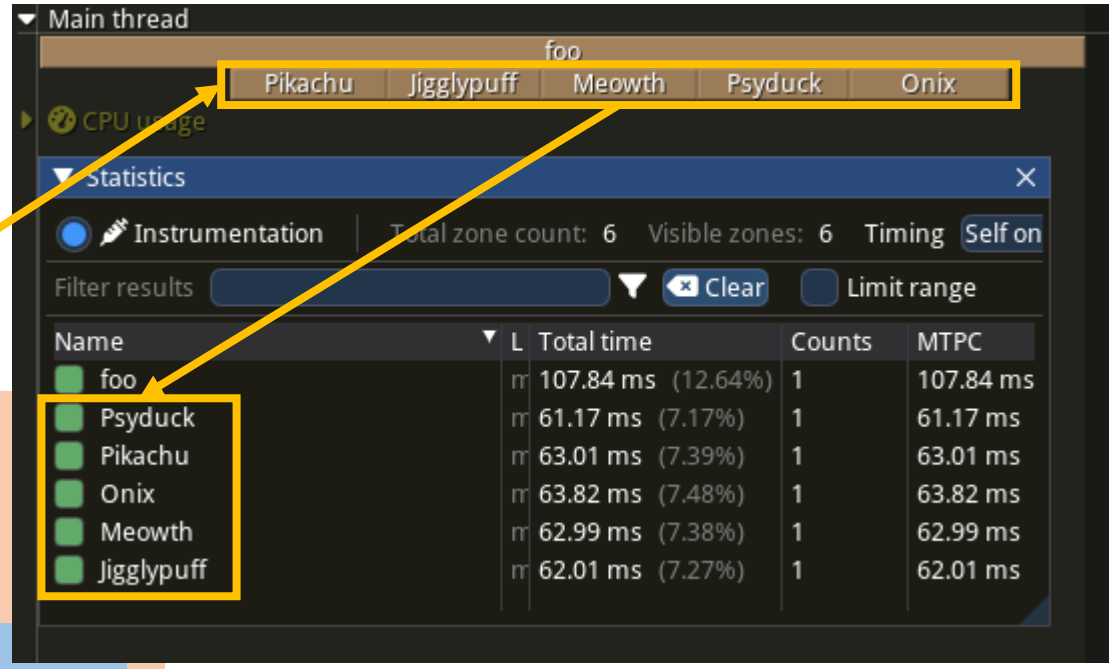
Tracy Zoning 401

```
int foo()
{
    ZoneScoped;
    Pokedex pokedex;
    for (auto& pokemon : pokedex)
    {
        ZoneScopedN("pokemon");
        ZoneName(pokemon.name, strlen(pokemon.name));
        ...
    }
}
```



Tracy Zoning 401

```
int foo()
{
    ZoneScoped;
    Pokedex pokedex;
    for (auto& pokemon : pokedex)
    {
        ZoneScopedN("pokemon");
        ZoneTransientN(pokezone, pokemon.name, true);
        ...
    }
}
```



Transient zones **may be necessary** when instrumenting a module that can be loaded **and unloaded** throughout the lifecycle of the host application (e.g., a plugin)

3.4.4 Transient zones

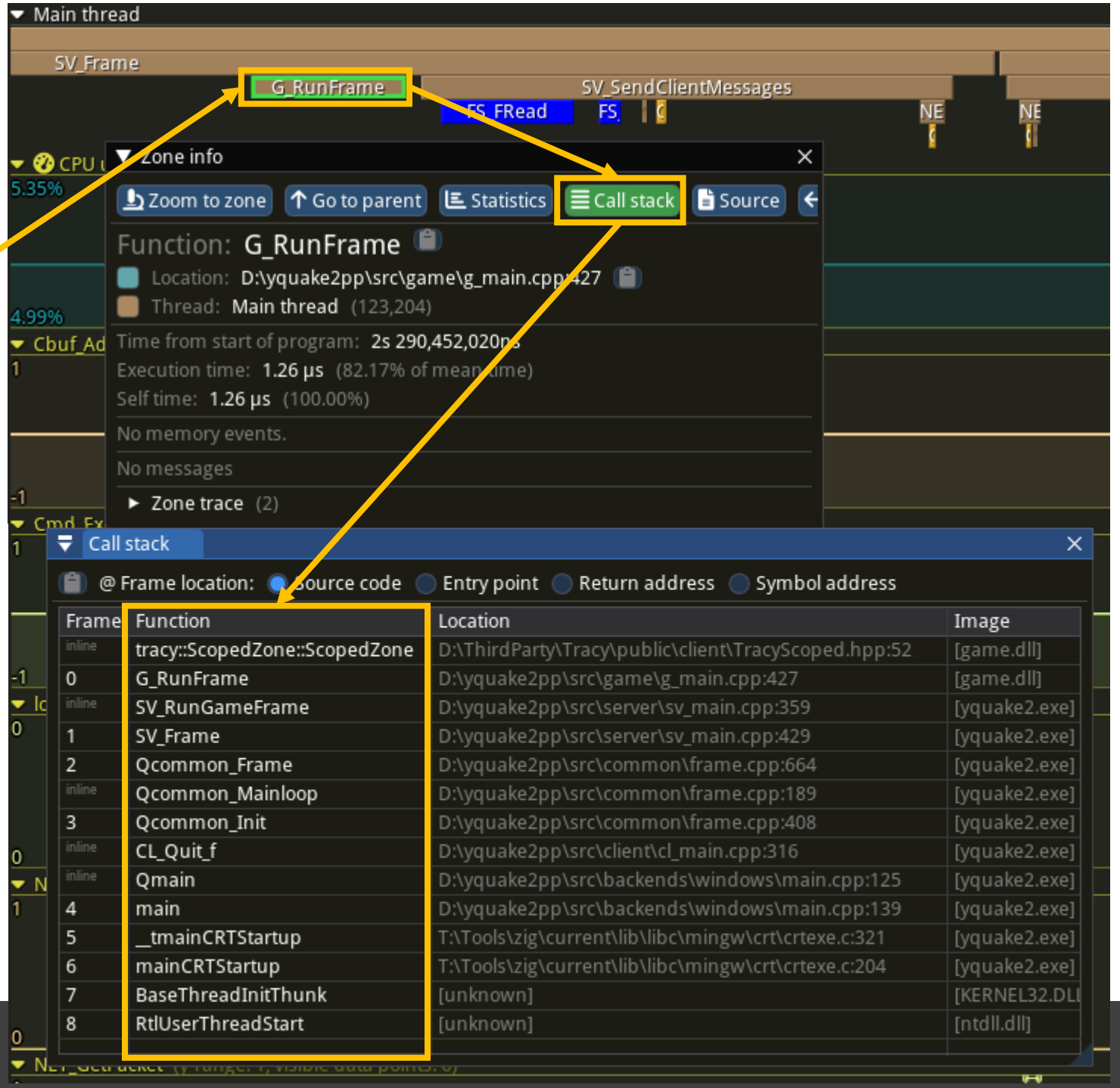
In order to prevent problems caused by unloadable code, described in section 3.1.1, transient zones copy the source location data to an on-heap buffer. This way, the requirement on the string literal data being accessible for the rest of the program lifetime is relaxed, at the cost of increased memory usage.

- **ZoneTransientN**
  - name a zone with **non-persistent string**
  - the **same zone** can have **multiple names**
  - will **show up in statistics!**
  - increased **runtime cost** to record zone

Tracy Zoning 501

```
g_main.cpp
game
421  /*
422   * Advances the world by 0.1 seconds
423   */
424  void
425  G_RunFrame(void)
426  {
427      ZoneScopedS(60);
428
429      int i;
430      edict_t *ent;
```

- ZoneScopedS(depth)
  - captures callstack
  - adds some overhead
  - useful for code exploration
  - max callstack depth: ~60
  - (enables selective sampling profiling)
- ZoneScoped[N|C|S]

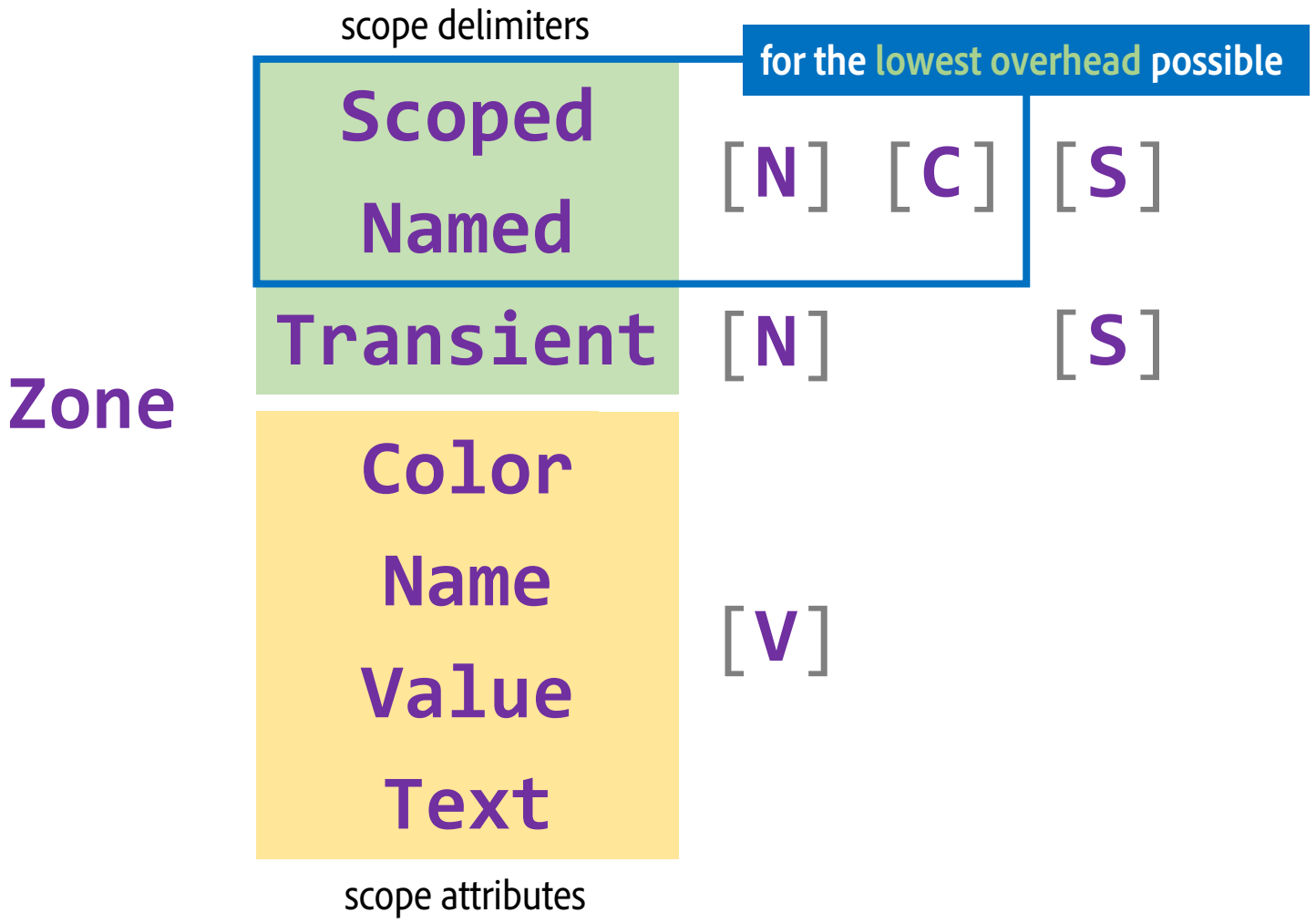


Callstack capture cost

Depth	x86	x64	ARM	ARM64
1	34 ns	98 ns	6.62 µs	6.63 µs
2	35 ns	150 ns	8.08 µs	8.25 µs
3	36 ns	168 ns	9.75 µs	10 µs
4	39 ns	190 ns	10.92 µs	11.58 µs
5	42 ns	206 ns	12.5 µs	13.33 µs
10	52 ns	306 ns	19.62 µs	21.71 µs
15	63 ns	415 ns	26.83 µs	30.13 µs
20	77 ns	531 ns	34.25 µs	38.71 µs
25	89 ns	630 ns	41.17 µs	47.17 µs
30	109 ns	735 ns	48.33 µs	55.63 µs
35	123 ns	843 ns	55.87 µs	64.09 µs
40	142 ns	950 ns	63.12 µs	72.59 µs
45	154 ns	1.05 µs	70.54 µs	81 µs
50	167 ns	1.16 µs	78 µs	89.5 µs
55	179 ns	1.26 µs	85.04 µs	98 µs
60	193 ns	1.37 µs	92.75 µs	106.59 µs

Tracy Manual: Section 3.11 – Collecting Timestamps

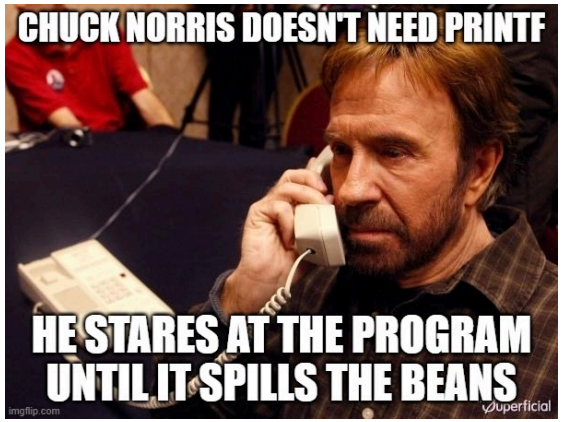
Recap (cheat sheet)



Suffixes:

- **N** : explicit name
- **C** : explicit color
- **S** : capture callstack
- **V** : explicit zone scope identifier
  (when using `ZoneNamed[N|C|S]`)
  (or `ZoneTransient[N|S]`)

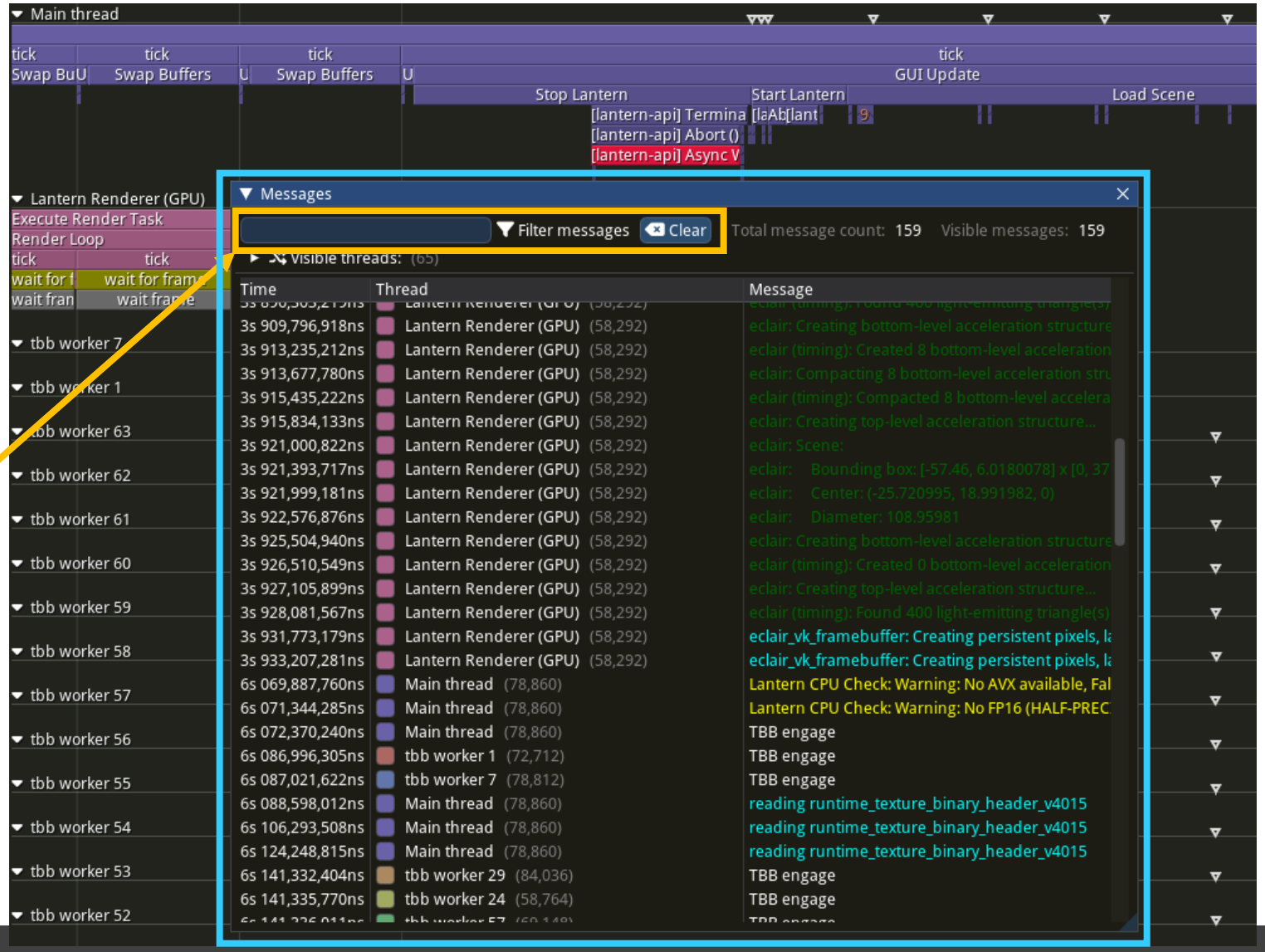
TracyMessage() is the new printf()



persistent string literal

TracyMessage[L | C | S]

- console replacement!
- time-aware!
- thread-aware!
- message colors!
- browsable log!
- filterable log!
- no thread sync needed!



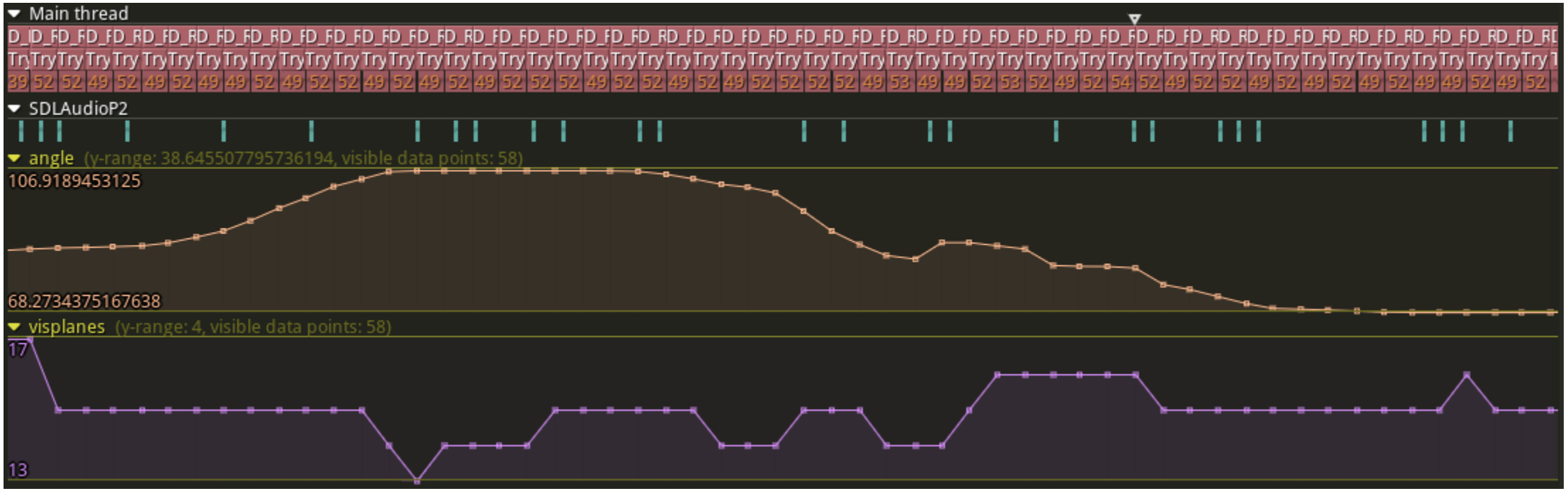
Plots

- `TracyPlot(name, value)`

  persistent **string literal**

  `int64`, `float` or `double`

plots are **shared** among threads
no per-thread plot segregation
but profiler **sorts on timestamp**



Plots

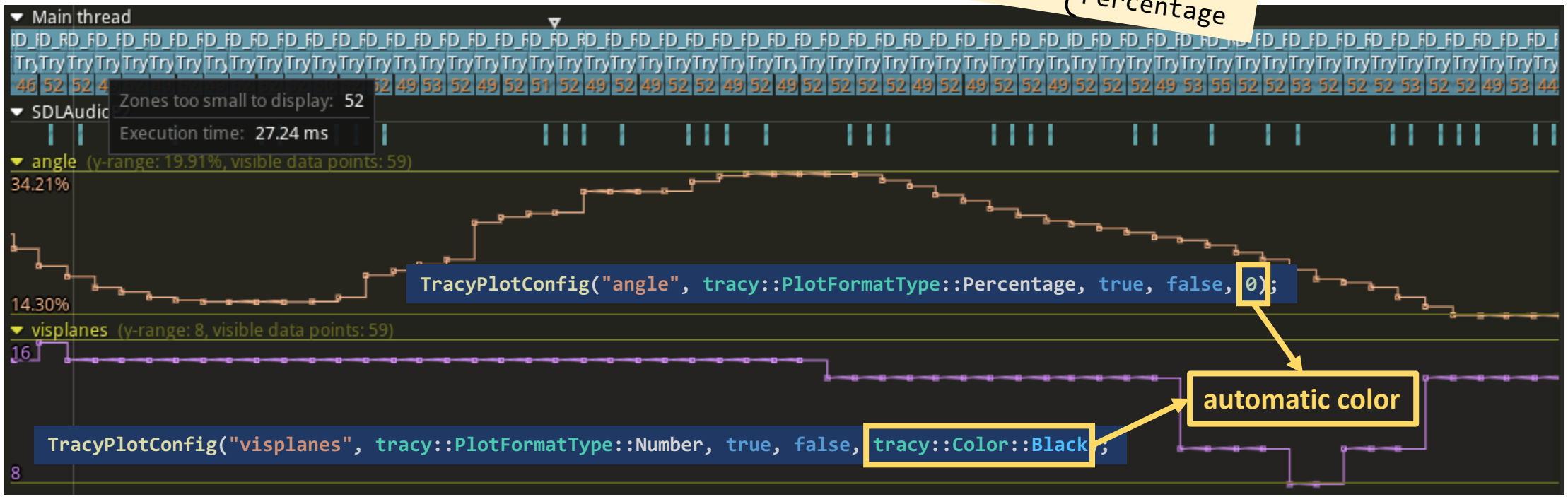
- `TracyPlotConfig(name, type, step, fill, color)`

`true`: stairstep
`false`: linear

`true`: shade area
`false`: outline only

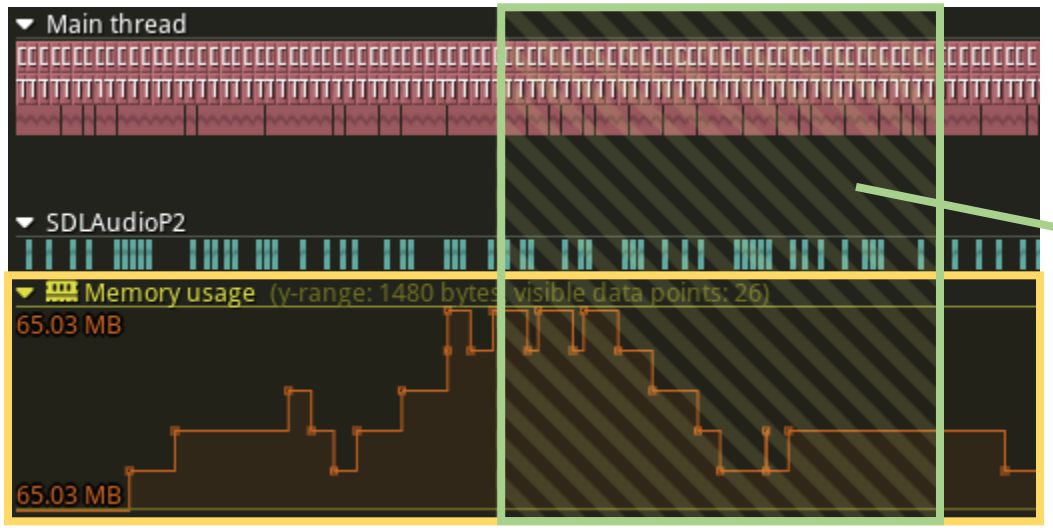
`tracy::Color::`

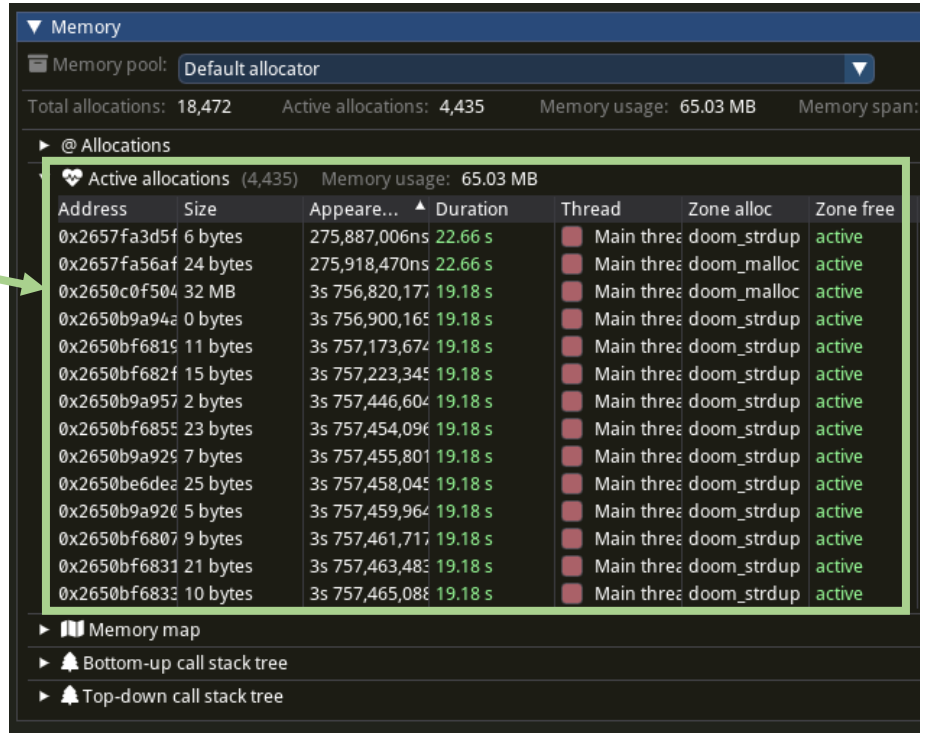
`tracy::PlotFormatType::` { Number, Memory, Percentage



Tracking Memory Allocations

- `TracyAlloc[N|S](ptr, size, ...)`
- `TracyFree[N|S](ptr, ...)`





TracyAlloc and TracyFree **won't** do memory **allocation**: it's merely for **tracking**

Lock instrumentation

```
std::mutex my_mutex;
```

```
TracyLockable(std::mutex, my_mutex);
```

and then

```
std::lock_guard<LockableBase(std::mutex)> lock (my_mutex);
```

There's also **TracySharedLockable[N]** (and **SharedLockableBase**) for reader-writer mutex

What's Next?

Tips & Tricks
(& Gotchas)

Study Cases
from Adobe

Explore
Together

&

Time for
Questions

GPU
instrumentation
(very briefly)

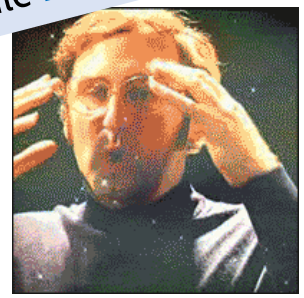
A Look at Tracy's
Sampling Mode

Tips & Tricks

Apple users...

...get a **real mouse***!

holding right button while moving pointer!



*or help fix Apple trackpad gestures in ImGui

Tips & Tricks

Instrument the "**usual suspects**"

• Idling	usually a good thing!
• Waits	implicit ones too: e.g. future.get()
• Sleeps	please, just don't…
• Lock acquisition (contention)	~~TracyLockable~~
• Memory operations	memcpy(), malloc() et al.
• Explicit system calls (or heavy API calls)	shader compilation
• Big-O activity (high-level)	sorting, searching, container manipulations, …
• I/O activity	file, network, logging, …
• Unpredictable events	interrupts, user input activity, spurious wake-ups, …

Tips & Tricks

Color zones **appropriately**

- Idling — usually a good thing!
- Waits — ***implicit*** ones too: e.g. future.get()
- Sleeps — better yet, avoid sleep() in production code!
- Lock acquisition (contention) — ~~TracyLockable~~
- Memory operations — memcpy(), malloc() et al.
- Explicit system calls (or heavy API calls) — shader compilation
- Big-O activity (high-level) — sorting, searching, container manipulations, …
- I/O activity — file, network, logging, …
- Unpredictable events — interrupts, user input activity, spurious wake-ups, …

Tips & Tricks

Color zones appropriately

- Idling — good thing!
- Waits — ...ture.get()
- Sle... — ...ion code!
- Lock... — ...ockable
- Mem... — ...() et al.
- Explic... — ...compilation
- Big-O ... — ...g, searching, container manipulations, ...
- I/O acti... — file, network, logging, ...
- Unpredi... ...events — interrupts, user input activity, spurious wake-ups, ...

Just to be clear...

Rely on Tracy's **AUTOMATIC** coloring **EVERYWHERE**

EXCEPT for these **KNOWN-TO-BE-PROBLEMATIC** cases

(they **usually** appear at the **bottom** of the **zone stack**)

pick **vibrant colors**: you want them to **pop up** in the profiler!

(Tracy's automatic coloring never chooses such saturated colors)

Tips & Tricks

Standardize the colors (don't think about them!)

Semantic Palette

```
enum Tier
{
    Idle   = tracy::Color::DimGray,
    Wait   = tracy::Color::Crimson,
    Sleep  = tracy::Color::Tomato,
    Lock   = tracy::Color::Coral,
    Memory = tracy::Color::Goldenrod,
    System = tracy::Color::Gold,
    BigO   = tracy::Color::LimeGreen,
    IO     = tracy::Color::Blue,
    Fickle = tracy::Color::Magenta,
    ...
};
```

X11 color names

(consider color blindness!)

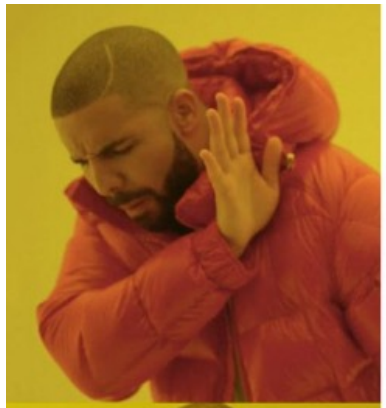
Tips & Tricks

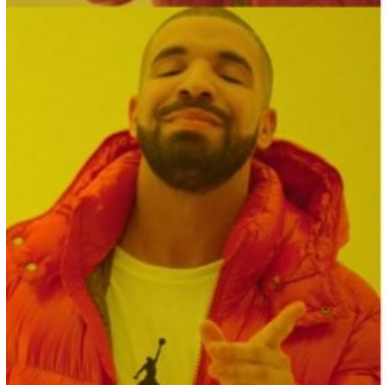
Name your **threads**!

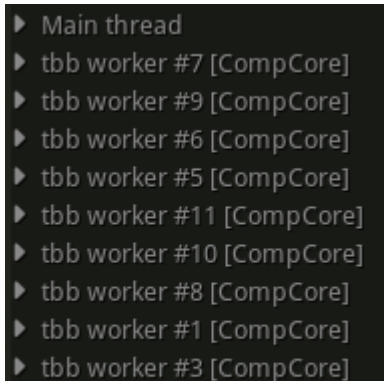
```
tracy::SetThreadName(“thread-name”)
```

TBB workers : use `task_scheduler_observer`

```
class tracy_tbb_observer : public oneapi::tbb::task_scheduler_observer {
    public:
    tracy_tbb_observer(tbb::task_arena& arena) : oneapi::tbb::task_scheduler_observer(arena) {
        observe(true); // activate the observer
    }
    void on_scheduler_entry(bool is_worker) override {
        if (is_worker) {
            int tid = tbb::this_task_arena::current_thread_index();
            tracy::SetThreadName(std::string("tbb worker #").append(std::to_string(tid)).c_str());
        }
    }
    void on_scheduler_exit(bool is_worker) override { }
};
```







Tips & Tricks

Consider **"siloing"** the **usual suspects**

example: don't call memcpy() directly… call some my_memcpy() which then calls memcpy()

```
void* my_memcpy(void* dst, const void* src, size_t size)
{
    ZoneScopedC(tracy::Color::Blue);
    return ::memcpy(dst, src, size);
}
```

unified point for instrumenting the usual suspects!

whole-program **symbol hijacking**

(e.g., to replace `malloc` without siloing across all process modules)

LD_PRELOAD

DYLD_INSERT_LIBRARIES

Microsoft Detours

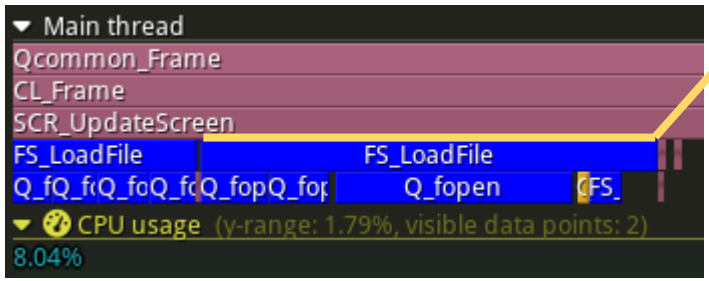
weak symbol linkage interposition

MSVC linker /alternatename

Tips & Tricks

Capture **callstack** on **"terminal"** zones for **"code reconnaissance"**

```
649  int
650  FS_LoadFile(char *path, void **buffer)
651  {
652      ZoneScopedCS(tracy::Color::Blue, 64);
653      ZoneText(path, strlen(path));
654
655      byte *buf; /* Buffer. */
656      int size; /* File size. */
657      fileHandle_t f; /* File handle. */
658
```



Call stack

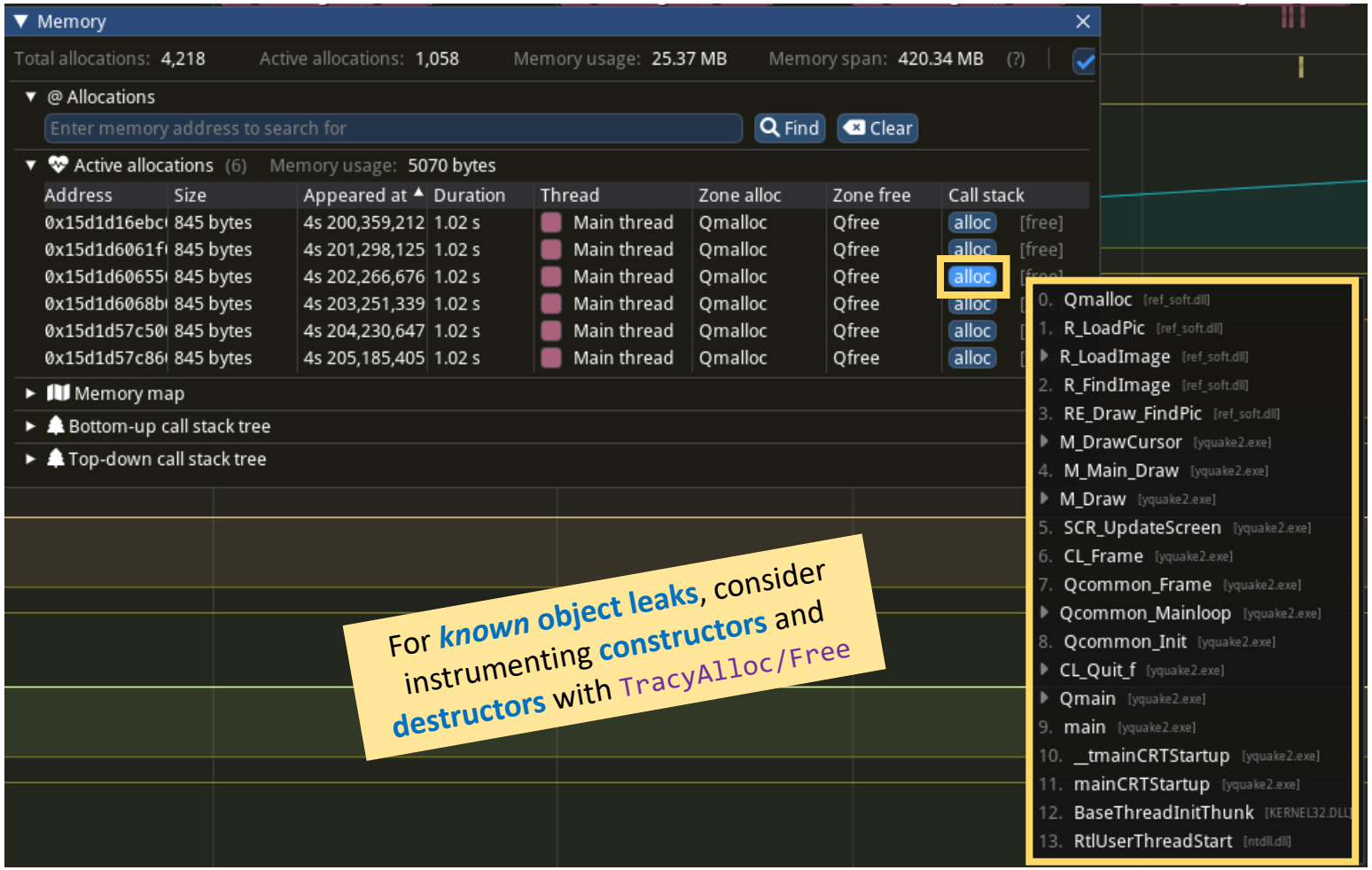
@ Frame location: Source code | Entry point | Return address | Symbol address

Frame	Function	Location	Image
inline	tracy::ScopedZone::ScopedZone	D:\yquake2pp\external\tracy\public\client\TracyScoped.hpp:52	[yquake2.exe]
0	FS_LoadFile	D:\yquake2pp\src\common\filesystem.cpp:652	[yquake2.exe]
1	LoadPCX	D:\yquake2pp\src\client\refresh\files\pcx.cpp:150	[ref_soft.dll]
inline	R_LoadImage	D:\yquake2pp\src\client\refresh\soft\sw_image.cpp:593	[ref_soft.dll]
2	R_FindImage	D:\yquake2pp\src\client\refresh\soft\sw_image.cpp:697	[ref_soft.dll]
3	RE_Draw_PicScaled	D:\yquake2pp\src\client\refresh\soft\sw_draw.cpp:392	[ref_soft.dll]
4	M_Main_Draw	D:\yquake2pp\src\client\menu\menu.cpp:621	[yquake2.exe]
inline	M_Draw	D:\yquake2pp\src\client\menu\menu.cpp:4973	[yquake2.exe]
5	SCR_UpdateScreen	D:\yquake2pp\src\client\cl_screen.cpp:1656	[yquake2.exe]
6	CL_Frame	D:\yquake2pp\src\client\cl_main.cpp:843	[yquake2.exe]
7	Qcommon_Frame	D:\yquake2pp\src\common\frame.cpp:680	[yquake2.exe]
inline	Qcommon_Mainloop	D:\yquake2pp\src\common\frame.cpp:189	[yquake2.exe]
8	Qcommon_Init	D:\yquake2pp\src\common\frame.cpp:408	[yquake2.exe]
inline	CL_Quit_f	D:\yquake2pp\src\client\cl_main.cpp:316	[yquake2.exe]
inline	Qmain	D:\yquake2pp\src\backends\windows\main.cpp:125	[yquake2.exe]
9	main	D:\yquake2pp\src\backends\windows\main.cpp:139	[yquake2.exe]
10	__tmainCRTStartup	T:\Tools\zig\current\lib\libc\mingw\crt\crtexe.c:321	[yquake2.exe]
11	mainCRTStartup	T:\Tools\zig\current\lib\libc\mingw\crt\crtexe.c:204	[yquake2.exe]
12	BaseThreadInitThunk	[unknown]	[KERNEL32.DLL]
13	RtlUserThreadStart	[unknown]	[ntdll.dll]

- the **"usual suspects"** are usually **leaf/terminal** calls
- they appear at the **bottom** of the **zone stack**
- ideal **candidates** for **code recon** via **callstack** inspection

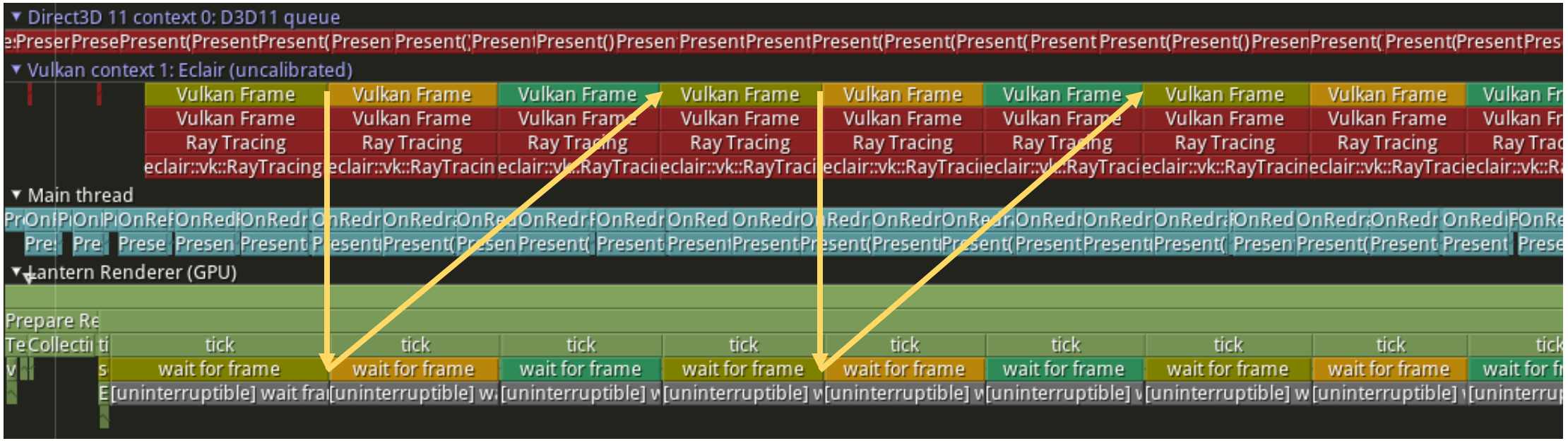
Tips & Tricks

Capture **callstack** on **allocation** to hunt down **leaks** : **TracyAlloc[N]S()**



Tips & Tricks

Use **color cycling** to visually **correlate** zones



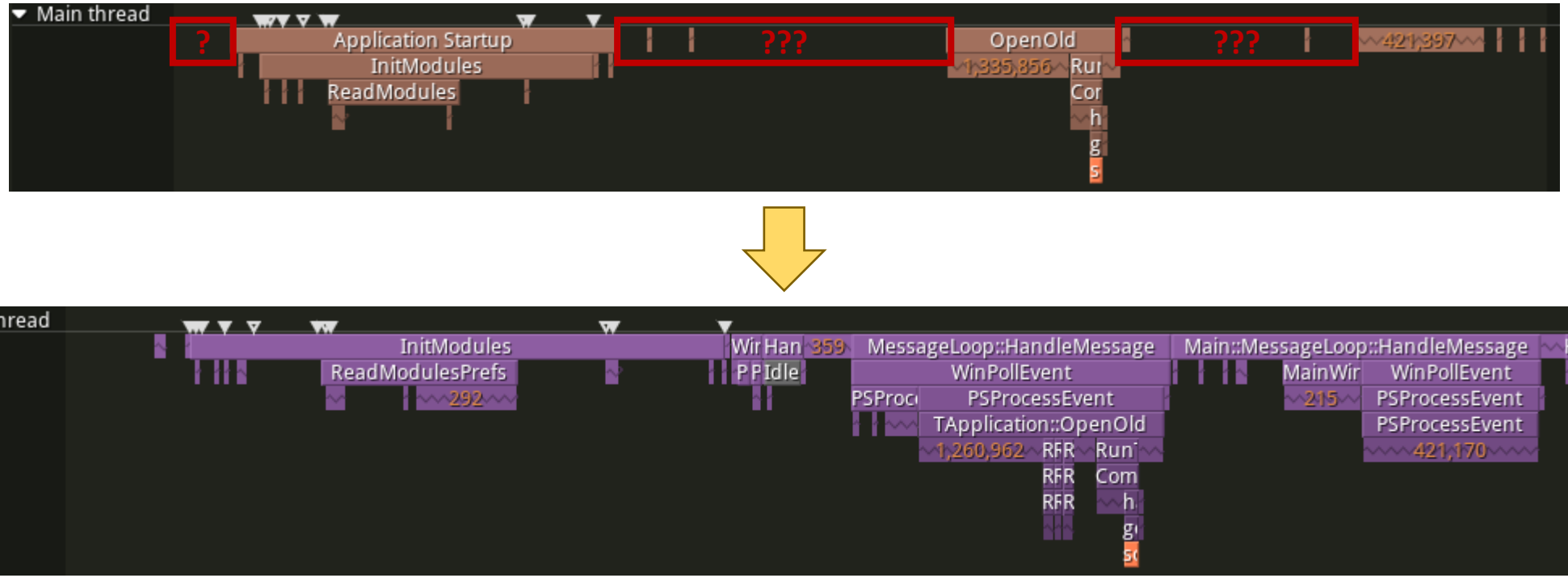
Tips & Tricks

Use TracyIsConnected() to branch on “expensive” tracing stuff

```
if (TracyIsConnected()) {
    // expensive stuff I want to see in the profiler
    // (now that I know the profiler is connected to the host app)
    // ...
    ZoneScopedS(60);    // let’s capture the callstack
    // ...
    ZoneText(...);
    // ...
    TracyMessage(...);
    // ...
}
```

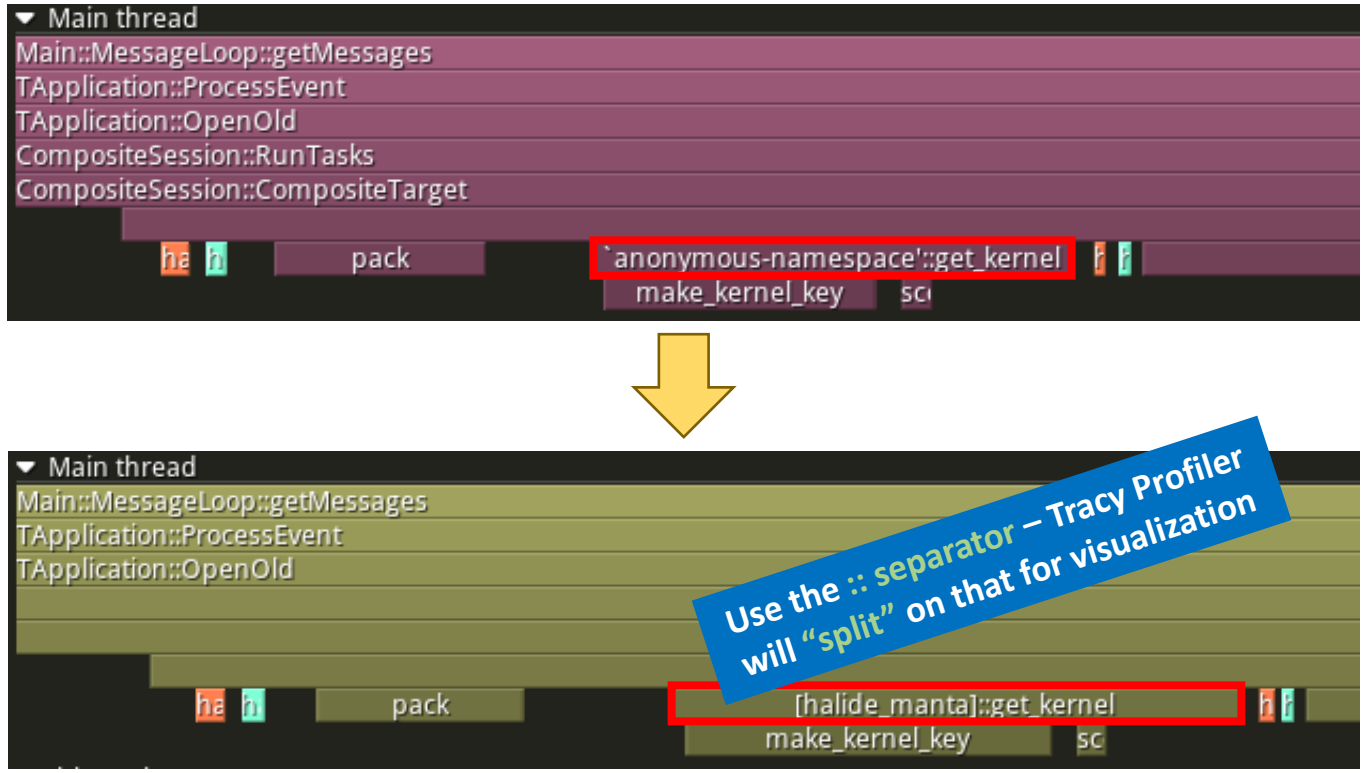
Tips & Tricks

Avoid "**empty gaps**" between zones (at least in the **top-most** level)

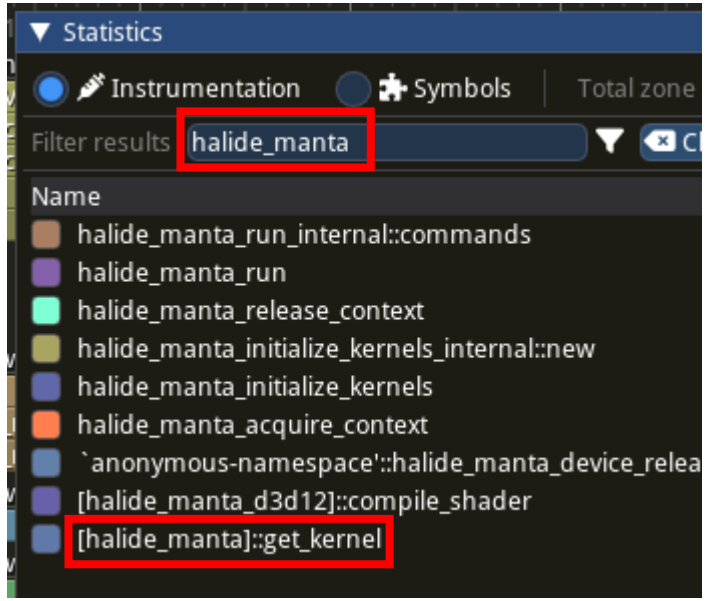


Tips & Tricks

Add **API prefix** to zone name when no prefix/namespace exists



useful for filtering in statistics window



Tips & Tricks

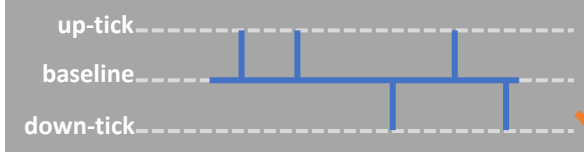
Use **tick signals** to visualize **critical activity**

- user input
- critical callback
- interrupt signals
- IPC signals
- completion signals
- wake-up signals

stairstep

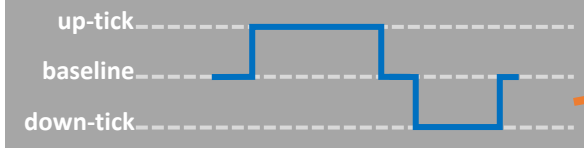
```
TracyPlotConfig(name, type, true, fill, color)
```

pulse (heartbeat) plots

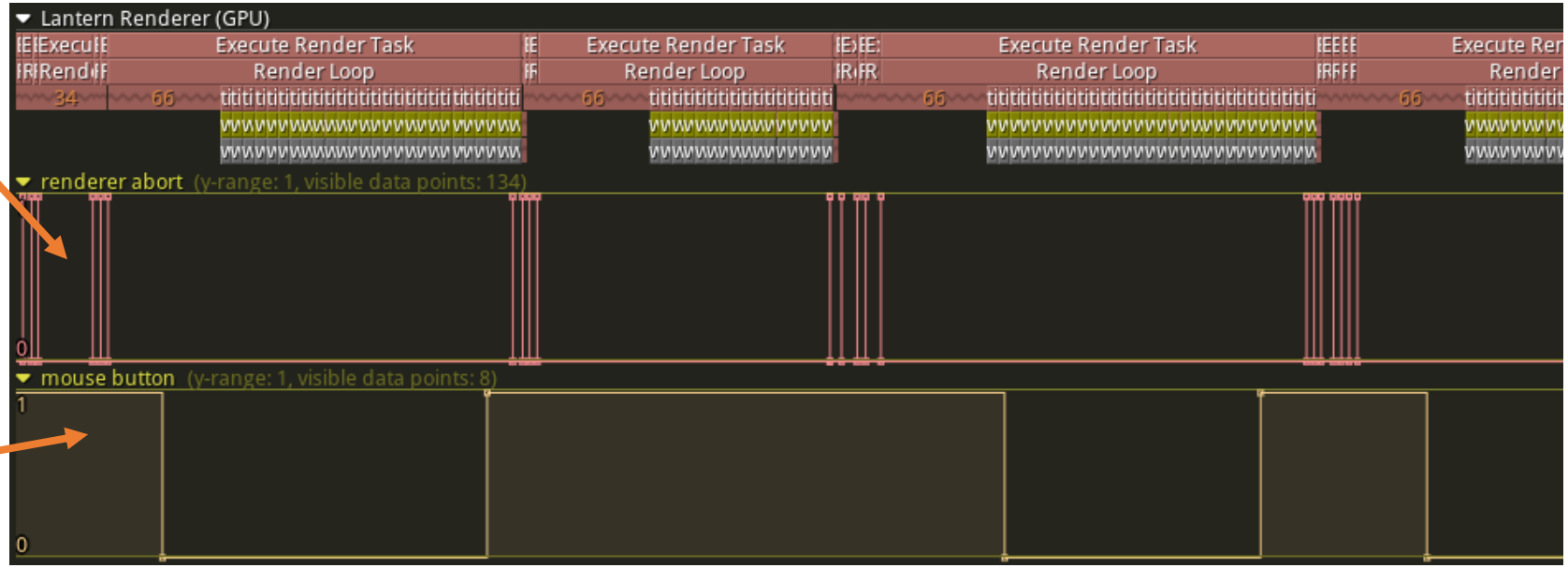



```
TracyPlot(name, tick);
TracyPlot(name, baseline);
```

clock (sustain) plots




```
TracyPlot(name, tick);
// ... activity ...
TracyPlot(name, baseline);
```

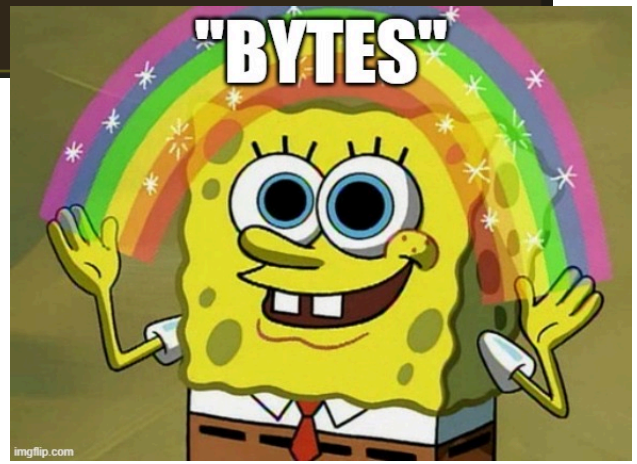


Tips & Tricks

Use TracyAlloc() and TracyFree() to track *anything* (objects, resources, handles, …)

bytes…

```
TracyAllocN(&frodo,   1, "party members");
TracyAllocN(&gandalf, 1, "party members");
TracyAllocN(&aragorn, 1, "party members");
// ...
TracyMessageL("[Gandalf] You shall not pass!");
TracyFreeN(&gandalf, "party members");
```



Tips & Tricks

Instrumenting lock **contention**

too intrusive...

```
std::mutex my_mutex;
```

```
TracyLockable(std::mutex, my_mutex);
```

and then

```
std::lock_guard<LockableBase(std::mutex)> lock (my_mutex);
```

do this siloing instead!

```
auto scoped_mutex_lock(std::mutex& m) {
    ZoneScopedC(tracy::Color::Coral);
    return std::unique_lock(m);
}
```

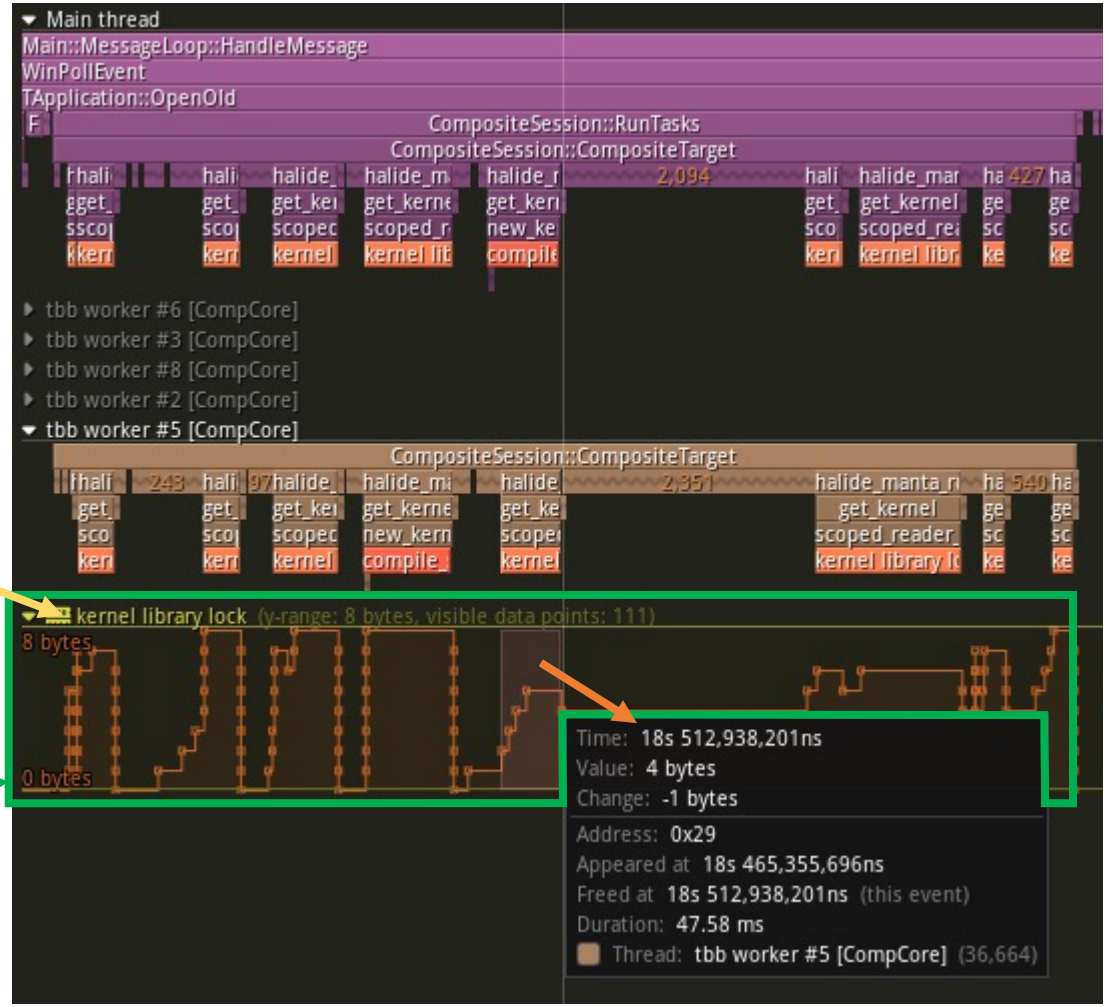
easily **adaptable to reader-writer mutex** as well
(std::shared_mutex and std::shared_lock)

Tips & Tricks

Measuring lock contention

```
static std::atomic<int> lock_id_generator = 0;

auto scoped_mutex_lock(std::mutex& m,
                       const char* name = "scoped_mutex_lock")
{
    if (m.try_lock())
    {
        // lucky! no contention!
        std::unique_lock lock (m, std::adopt_lock);
        return lock;
    }
    // uh-oh... possible contention...
    ZoneTransientN(mutex_lock_scope, name, true);
    ZoneColorV(mutex_lock_scope, tracy::Color::Coral);
    auto token = reinterpret_cast<void*>(++lock_id_generator);
    TracyAllocN(token, 1, name);
    std::unique_lock lock (m);
    TracyFreeN(token, name);
    return lock;
}
```

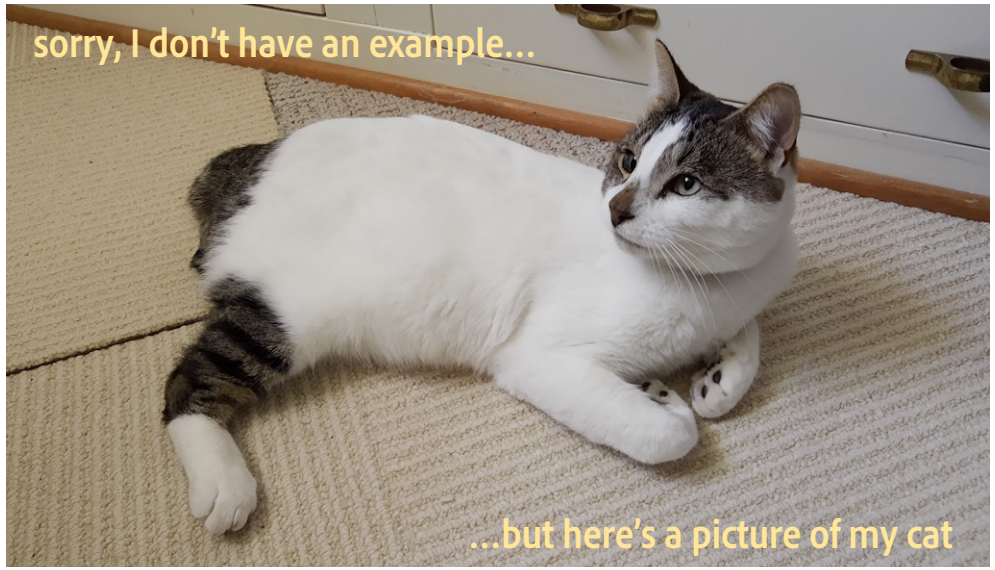


Tips & Tricks

Have an in-app **GUI** element to **launch & connect** the Tracy profiler

David Farrell's pro-tip!

Team must agree on a **common location** for the Tracy profiler executable

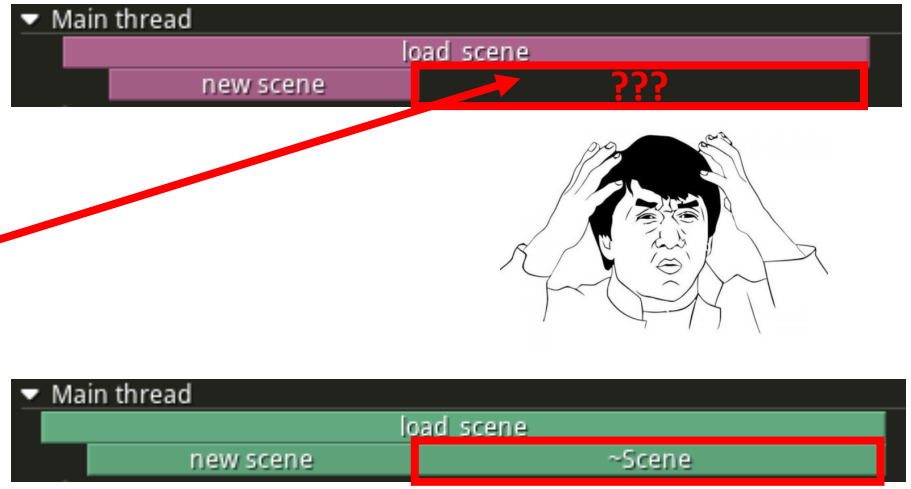


Gotchas

Unexplainable "tail" gaps:

- Implicit **deferred execution** (e.g. smart-pointer deleting an object)
- Instrument expensive **destructors**
- or **explicitly release** smart objects at the end of a scope

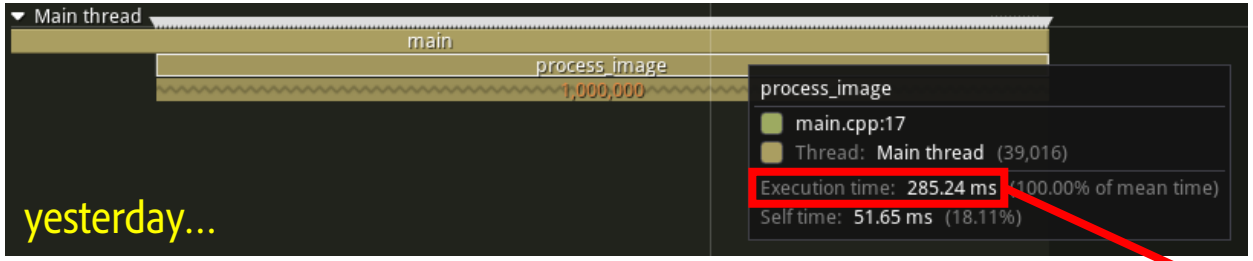
```cpp
bool Viewport::load_scene(const char* filename)
{
    ZoneScoped;
    // ...
    std::shared_ptr<Scene> new_scene;
    {
        ZoneScopedN("new scene");
        new_scene = std::make_shared<Scene>(filename);
        if (!new_scene) return false;
    }
    m_scene = new_scene;
    return true;
}
```

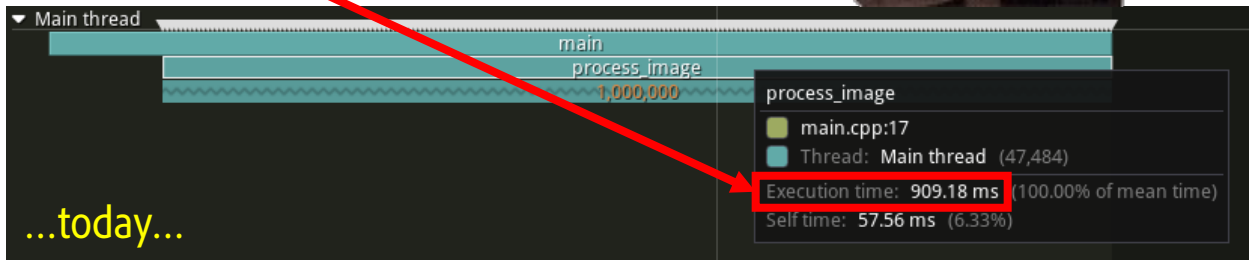


Gotchas

Zones are **suddenly** taking a lot **more time**

- Have you recently switched to a **Debug build**?
- Only profile **Release builds** (preferably **detached from debugger**)
- Run profiler tool on a **different machine**
- Have you since started **capturing callstacks** (too frequently)?





Gotchas

Beware **excessive** instrumentation!

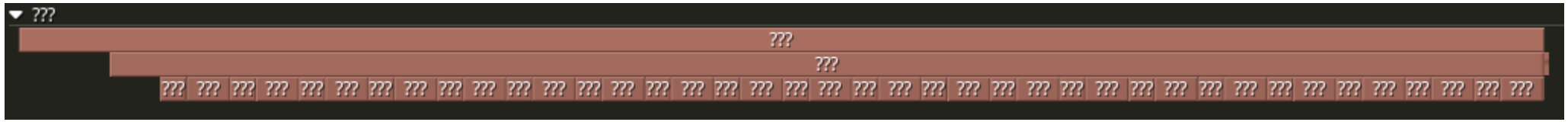
- hot **inner-loops**, small/inlined routines called frequently, …
- **Profiling *skew***: when profiling itself starts affecting performance
- May also affect **compiler optimizations**!

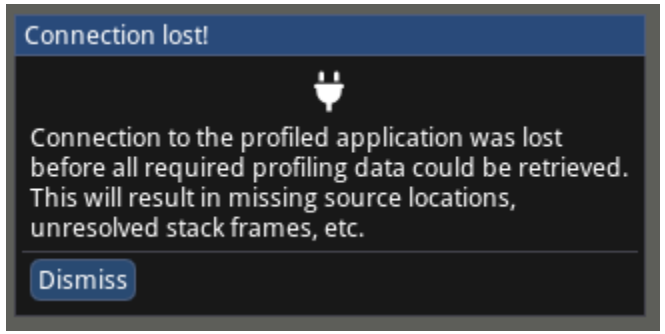
skewed profiling is still **useful**, but requires a **different mindset**

Gotchas

Profiler **not showing the name** of some zones: getting **???** instead

- just **wait** a little bit more (host application may not have sent this data yet)
- the host application may have **crashed** or terminated abruptly
- ensure zone name is a **persistent string**, or use **ZoneTransient()**
- if zone belongs to **plugin code**, make sure the plugin has **not been unloaded**

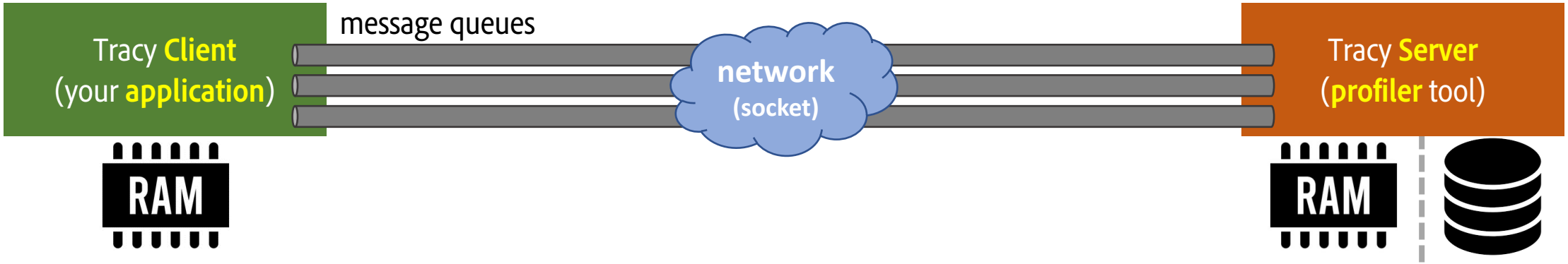




Gotchas

High RAM Usage

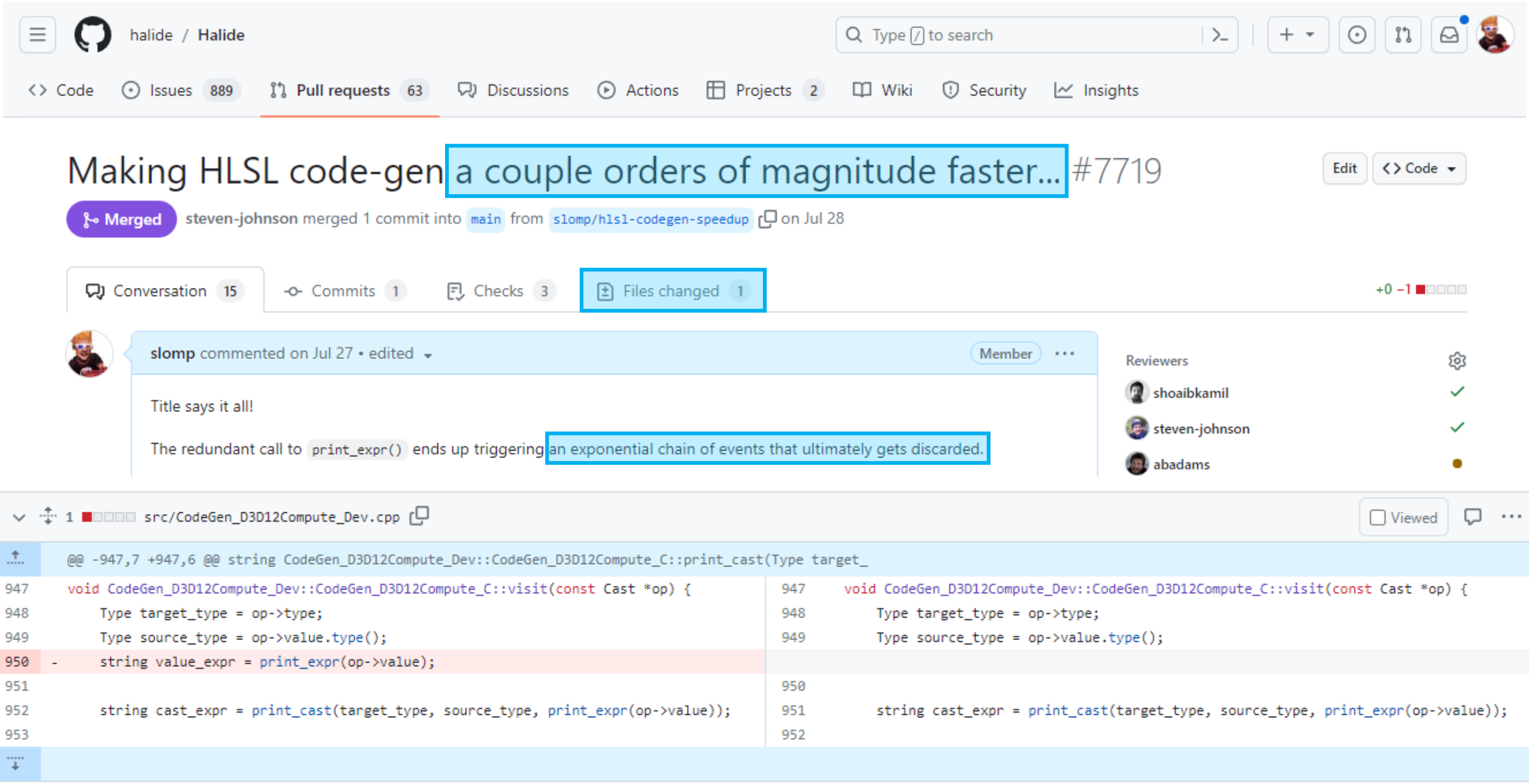
- Tracy **never drops** events! (there's no circular/LRU event queue)
- If profiler is **not connected**, application **memory keeps growing**!
- If profiler is running on the **same machine**, it is **competing for RAM**



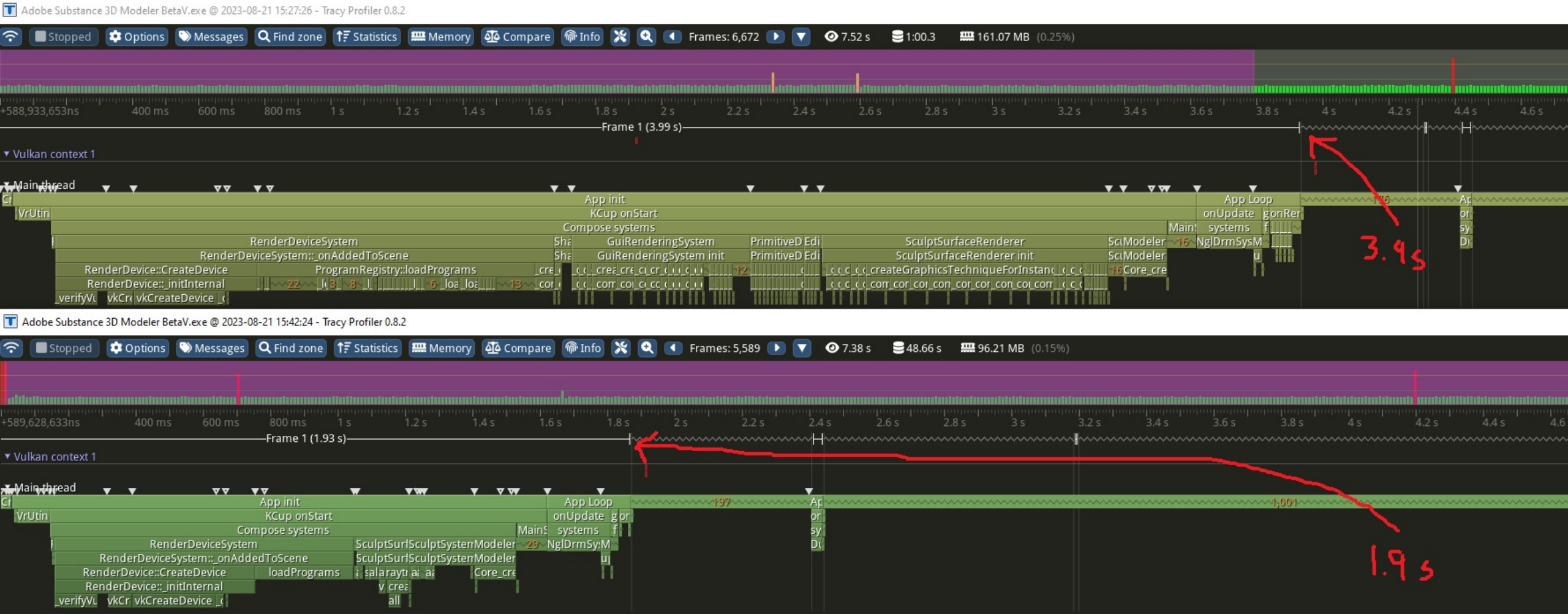
- #define TRACY_ON_DEMAND to delay producer until profiler gets connected
- Use **capture** utility to "**sink**" Tracy events to **disk**
- Consider running the profiler GUI on a **different machine**

Study Cases

- [Photoshop] Win32 API Detours
- [Photoshop] Multi-threading opportunity
- [Photoshop] Eliminating redundant computations
- [Photoshop] Addressing lock contention
- [Halide] 100x faster HLSL code generation!
- [Halide] Yeah… std::stringstream is kinda slow…
- [Modeler] Improved parallel shader compilation
- [Éclair] Avoiding GPU<->CPU copies with GPU interop


halide / Halide
Type / to search
Code
Issues 889
Pull requests 63
Discussions
Actions
Projects 2
Wiki
Security
Insights
Making HLSL code-gen a couple orders of magnitude faster... #7719
Edit
Code
Merged
steven-johnson merged 1 commit into main from slomp/hlsl-codegen-speedup on Jul 28
Conversation 15
Commits 1
Checks 3
Files changed 1
+0 −1
slomp commented on Jul 27 • edited
Member
Title says it all!
The redundant call to print_expr() ends up triggering an exponential chain of events that ultimately gets discarded.
Reviewers
shoaibkamil
steven-johnson
abadams
1 src/CodeGen_D3D12Compute_Dev.cpp
Viewed
@@ -947,7 +947,6 @@ string CodeGen_D3D12Compute_Dev::CodeGen_D3D12Compute_C::print_cast(Type target_
947 void CodeGen_D3D12Compute_Dev::CodeGen_D3D12Compute_C::visit(const Cast *op) {
948 Type target_type = op->type;
949 Type source_type = op->value.type();
950 - string value_expr = print_expr(op->value);
951
952 string cast_expr = print_cast(target_type, source_type, print_expr(op->value));
953
947 void CodeGen_D3D12Compute_Dev::CodeGen_D3D12Compute_C::visit(const Cast *op) {
948 Type target_type = op->type;
949 Type source_type = op->value.type();
950
951 string cast_expr = print_cast(target_type, source_type, print_expr(op->value));
952

Recent story (Adobe Substance 3D Modeler)



A Taste of Tracy's Sampling Mode

TracyClient.cpp is all you need!

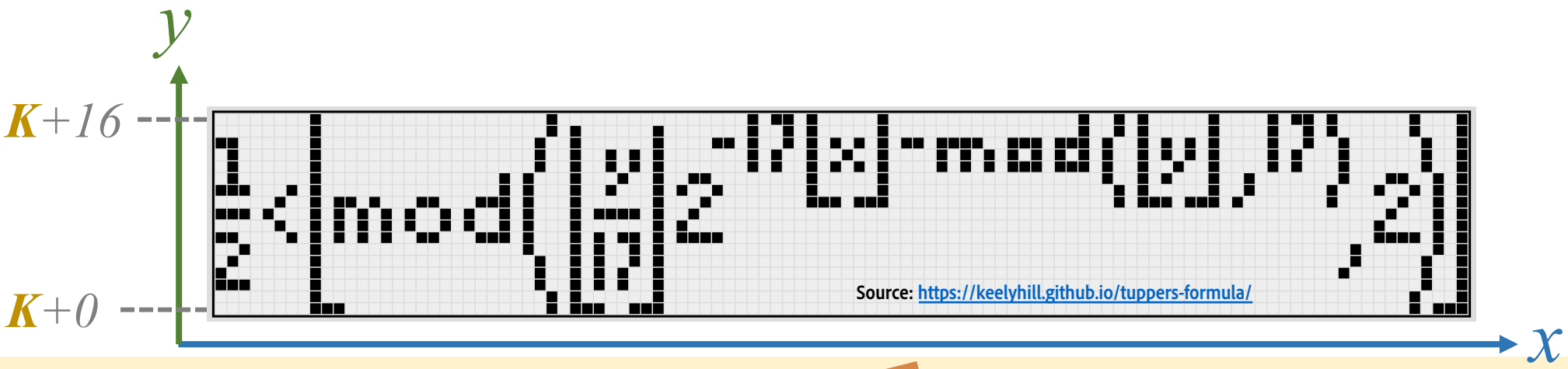
then launch your app with **admin privileges**
and connect it to the Tracy Profiler

Manual instrumentation is **not necessary**!
(but you can have **both** coexist **together**!)

A Taste of Tracy's Sampling Mode

Tupper's Self-Referential Formula

$$\frac{1}{2} < \left\lfloor \mathrm{mod}\left(\left\lfloor \frac{y}{17} \right\rfloor 2^{-17\lfloor x \rfloor - \mathrm{mod}(\lfloor y \rfloor, 17)}, 2 \right) \right\rfloor,$$



K =
485845063618971342358209596249420204458140058798324454948309308506193470470880928450644769865524364849997247024915119110411605739177407856919754326571855442057210445735883681829823754139634338225199452191651284348332905131103[illegible]5024137587652392648746133949068701305622958132194811136853395355652908500238750928568926945559742815463865107300491067230589335860525440966[illegible]126534936364395712556569593681518433485760526694016125126695142155053955451915378545752575659074054015792900176596796548006442782913148854825991472124850635268663047630 0

544 digits

A New Speedrun Category!

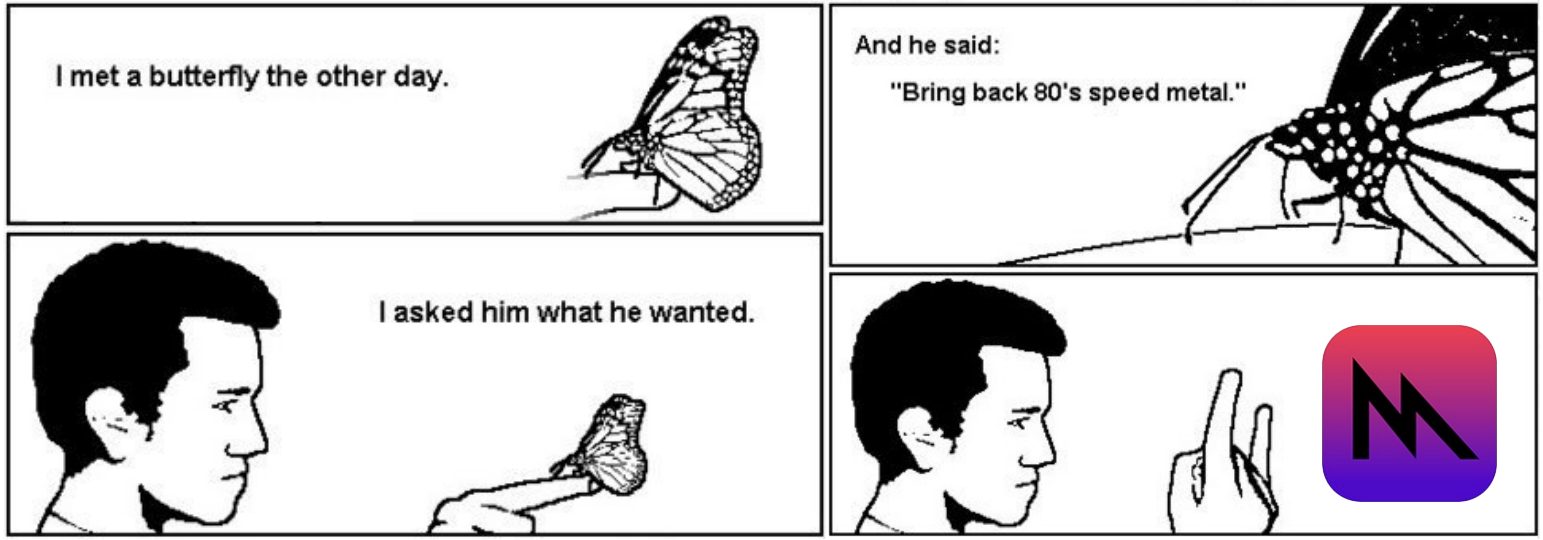
Quake 2 Demo First Frame Any%

World Record: **Marcos Slomp** **1.32s**

Known Issues

macOS/iOS Limitations

- Profiler may get **interrupted** when **application finishes**
  - It's Apple, not Tracy! Ongoing workaround discussion: https://github.com/wolfpld/tracy/issues/8
- Vertical scrollbar missing on main profiler window
  - It's ImGUI, not Tracy! https://github.com/wolfpld/tracy/issues/631
- No GPU **Metal API instrumentation** yet
  - I'm working on it!



Known Issues

Program **hangs** when calling **exit()**

- Try **TRACY_MANUAL_LIFETIME**
- or just **don't call exit()** or similar :-)

Known Issues

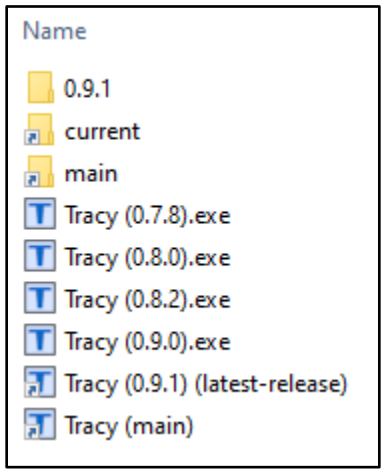
Tracy **capture files** are **not backwards compatible**

- Try using the **update** utility to convert trace files

download pre-built binary (Windows-only):
https://github.com/wolfpld/tracy/releases

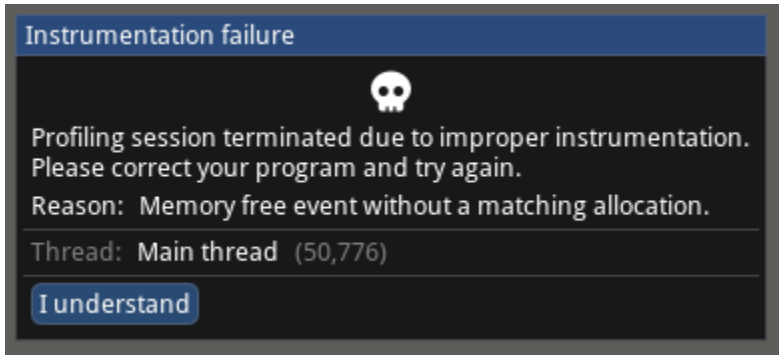
or build it yourself:
<tracy-repo>/update/build

- or **keep older versions** of Tracy around



Known Issues

TracyFree[N|S]() may **disconnect** profiler if **no matching allocation** exists



- Workaround:
  - force **m_ignoreMemFreeFaults** to **true** in **TracyWorker.cpp**
  - then make sure to **rebuild** the profiler tool

Known Issues

My program **won't connect** to the profiler

- may be the case for **short-lived** programs
- … that said, you may #define TRACY_NO_EXIT in the host program (client) to **prevent** it from exiting until a **connection** to the profiler is established

More Limitations

2.1.8 Limitations

When using Tracy Profiler, keep in mind the following requirements:

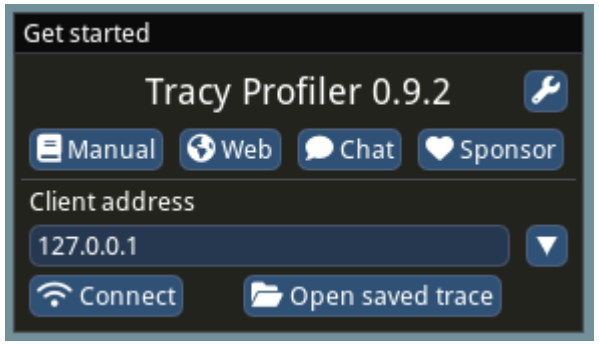
- The application may use each lock in no more than 64 unique threads.
- There can be no more than 65534 unique source locations[16]. This number is further split in half between native code source locations and dynamic source locations (for example, when Lua instrumentation is used).
- If there are recursive zones at any point in a zone stack, each unique zone source location should not appear more than 255 times.
- Profiling session cannot be longer than 1.6 days (2^{47} ns). This also includes on-demand sessions.
- No more than 4 billion (2^{32}) memory free events may be recorded.
- No more than 16 million (2^{24}) unique call stacks can be captured.

also: **callstack depth** limited to ***about* 60** stack frames

The following conditions also need to apply but don't trouble yourself with them too much. You would probably already know if you'd be breaking any.

- Only little-endian CPUs are supported.
- Virtual address space must be limited to 48 bits.
- Tracy server requires CPU which can handle misaligned memory accesses.

Help



RTTM: Read the Tracy Manual (it's ***really*** **good**!)

tracy.pdf (grab it from https://github.com/wolfpld/tracy/releases)

Discord server

https://discord.gg/pk78auc

C++ at Adobe!

developer.adobe.com/cpp

- Careers
- Events
- Training
- Blog Posts

Adobe

research.adobe.com

- Careers
- Internships
- Events
- Blog Posts

RESEARCH ENGINEERING & DESIGN

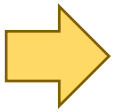
Tips & Tricks

Instrumenting **dynamic dispatch** (e.g., virtual calls)

Need to **instrument *all* overrides**…

Consider using **static delegate** interface to call the **virtual** method

```
class Shape
{
    public:
        virtual float area();
};
```



```
class Shape
{
    public:
        float area() {
            ZoneScoped;
            return calculate_area();
        }
    protected:
        virtual float calculate_area();
};
```

Tips & Tricks

Begin/End (and track) **zone** on **different scopes** or **across threads**

e.g.: task starts in main thread, moves between worker threads, and gets retired on a reclaimer thread

- A few **palliative** workarounds:
  - Use `FrameMarkStart` and `FrameMarkEnd`
  - Use “matching” `TracyMessage`’s
  - Use clock/sustain `TracyPlot`
  - Assign unique `ZoneColor` to each task

- My recommendation: use `TracyAlloc` and `TracyFree`
  (it’s basically a **stackable clock/sustain plot**!)

Tracy C API

Closing an outer zone inside an inner zone is bad!

- Tracy C-style API:
  - **explicit** scoping: must **store/carry** scope context object around
  - `#include <TracyC.h>`
  - `struct TracyCZoneCtx`
  - `TracyCZone[N|C|S](ctx, …, active)` : start/begin a zone
  - `TracyCZoneEnd(ctx)` : zone **MUST end** on the **SAME thread** that started it!
  - `TracyCZoneColor(ctx, …)`
  - `TracyCZoneValue(ctx, …)`
  - `TracyCZoneText(ctx, …)`
  - `TracyCZoneName(ctx, …)`

scoped zones are almost always **enough**
(and **intuitive**, and have **less overhead**)