

23

Tracy: A Profiler You Don't Want to Miss

MARCOS SLOMP



Cppcon
The C++ Conference

20
23



October 01 - 06

Tracy: a profiler you don't want to miss

Marcos Slomp

Senior Research Engineer II



Cppcon

The C++ Conference

2023 
October 01-06
Aurora, Colorado, USA



RESEARCH ENGINEERING & DESIGN

Adobe

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



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**



Kudos to Bartosz Taudul
creator and maintainer of Tracy



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



Kudos to David Farrell  for demonstrating Tracy's potential to me!



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>

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
Xpedite	https://github.com/morganstanley/Xpedite
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

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



Mathieu Ropert

20
21 | 
October 24-29

<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”*



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

- Bjarne Stroustrup
CppCon 2023 Opening Keynote
(literally three days ago!)

Closing the Profiling Zoo

Tracing

program execution logging

- function calls (+ args & return)
- system events
- state changes
- debug/warning/error messages

mostly for debugging: not necessarily concerned with performance

Sampling

callstack snapshot probing

- system-wide
- automatic (regular intervals)
- OS kernel events
- hardware counters

issues: information overload; sampling overhead; constrained by sampling frequency

Instrumentation

code annotations (markup)

holistic (automatic)
issues: information overload; profiling overhead (skew)

selective (manual)

Analysis

pre-recorded? | real-time?
log files? | command-line tool? | GUI?, ...

An **interactive**, **responsive** and **visual** tool transforms the profiling **experience!**

Closing the Profiling Zoo



Tracy Profiler GUI



Frame Info



GPU Timeline (per “device”)

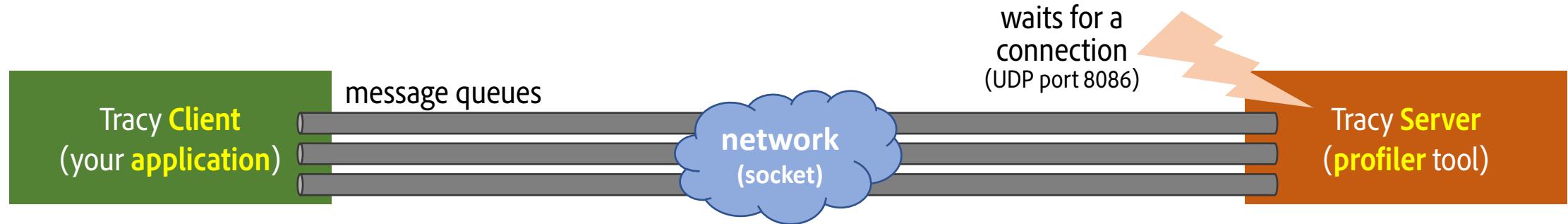


CPU Timeline (per-thread)

Custom Plots & Allocation Trackers



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

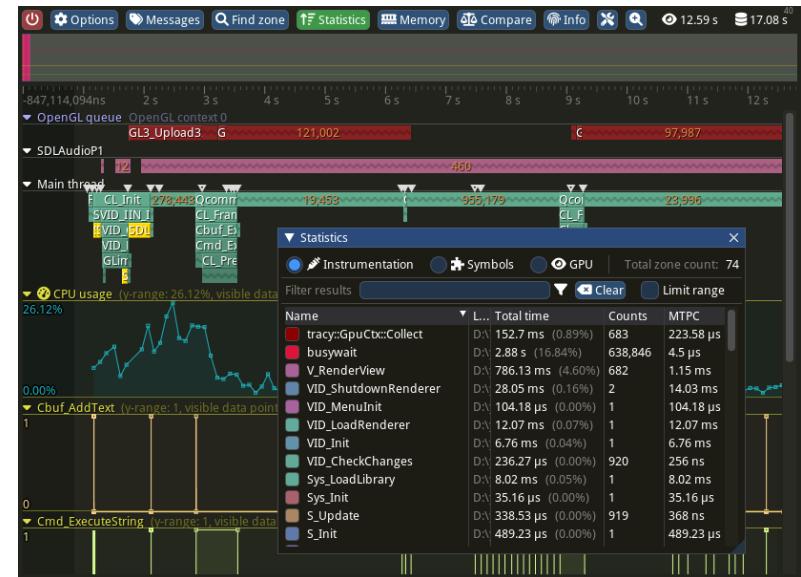
Quick integration (the *YOLO* way):

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

anywhere where needed

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

in one of the source files of your project

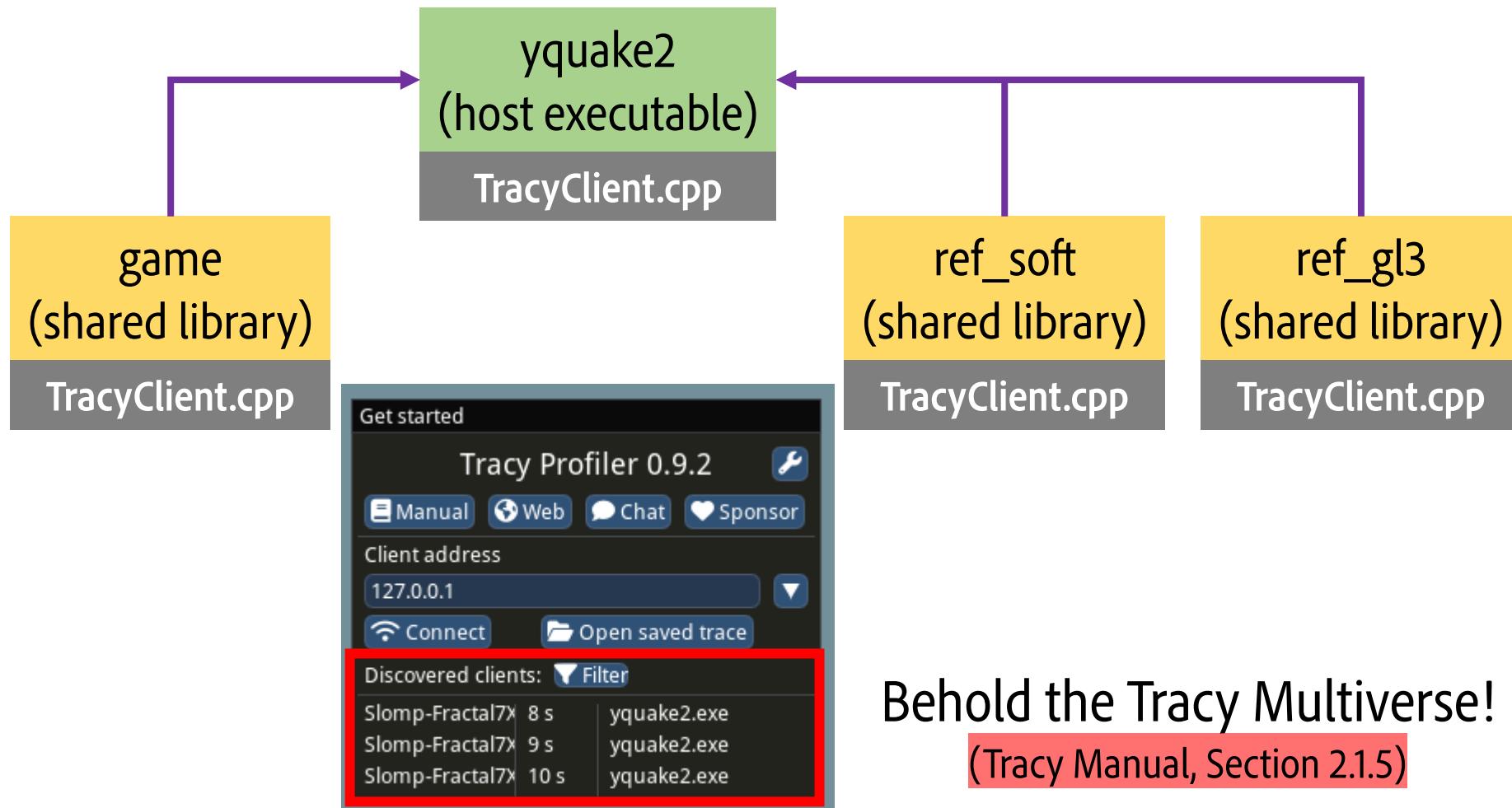


download pre-built binary (**Windows-only**):

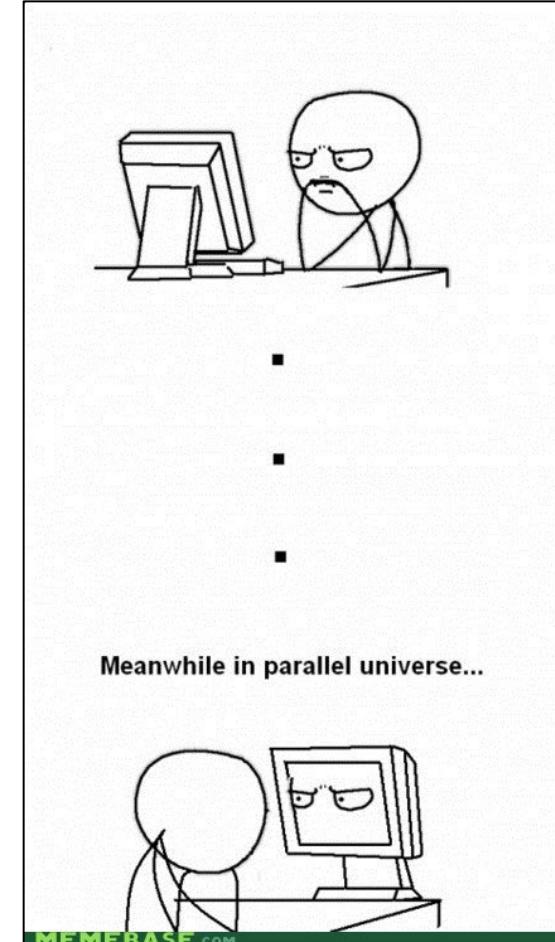
<https://github.com/wolfpld/tracy/releases>

or **build it yourself!**

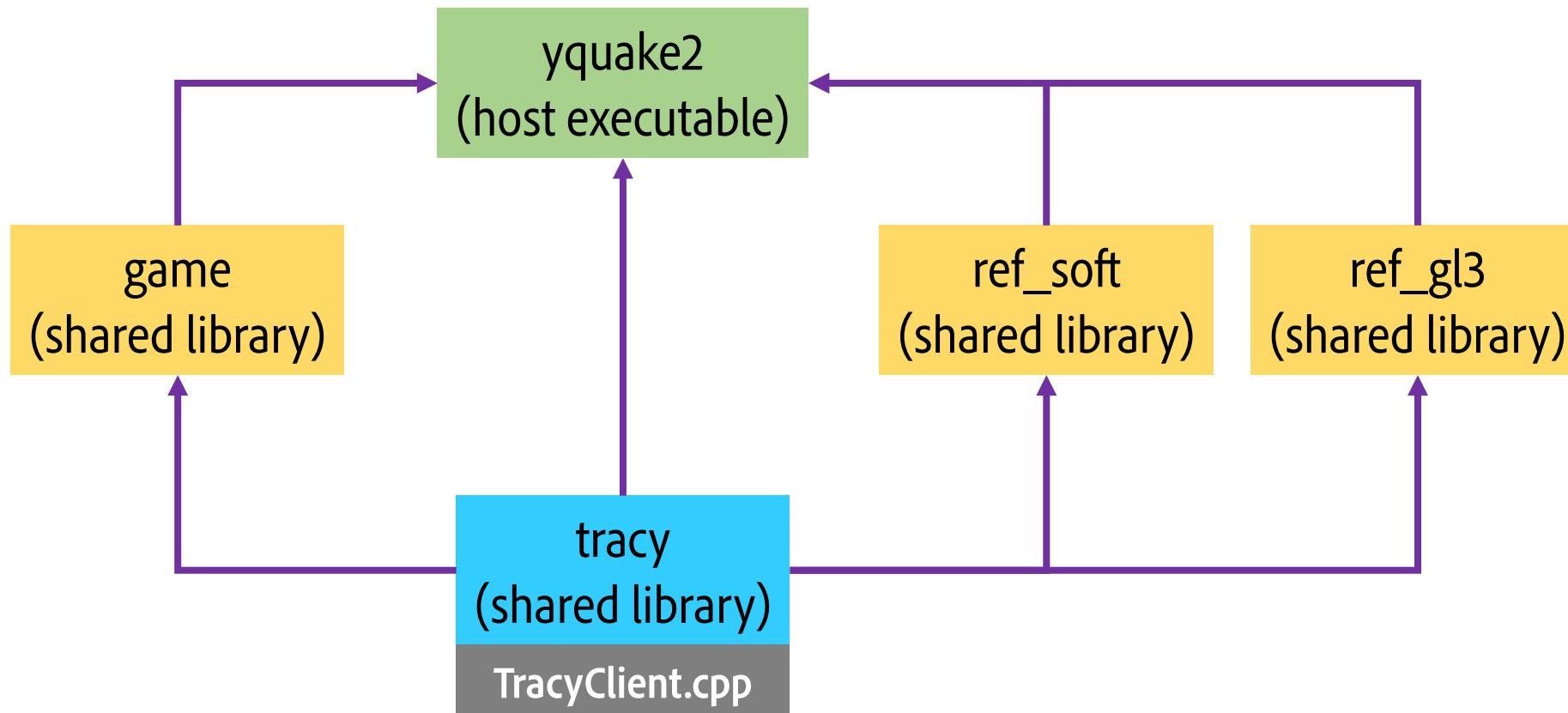
Tracy integration across shared libraries



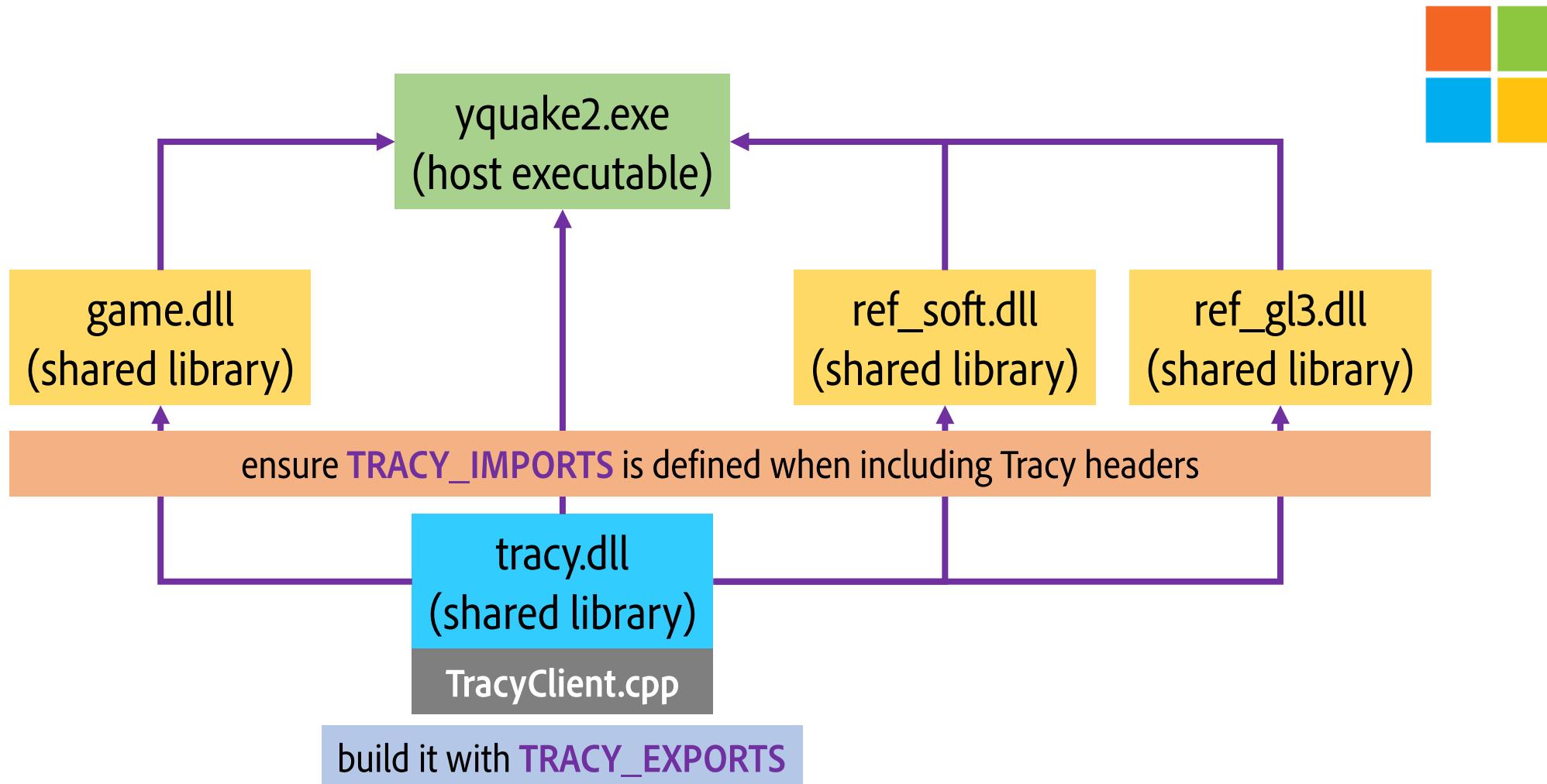
Behold the Tracy Multiverse!
(Tracy Manual, Section 2.1.5)



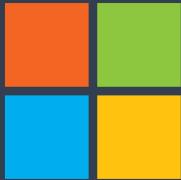
Tracy integration across shared libraries



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 -j[N]`

Tracy-release



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

2. Build with **make**:
`<tracy-repo>/profiler/build/unix $ make -j[N]`

Tracy-release

Zoning...

{ this is a zone scope (RAII) }

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



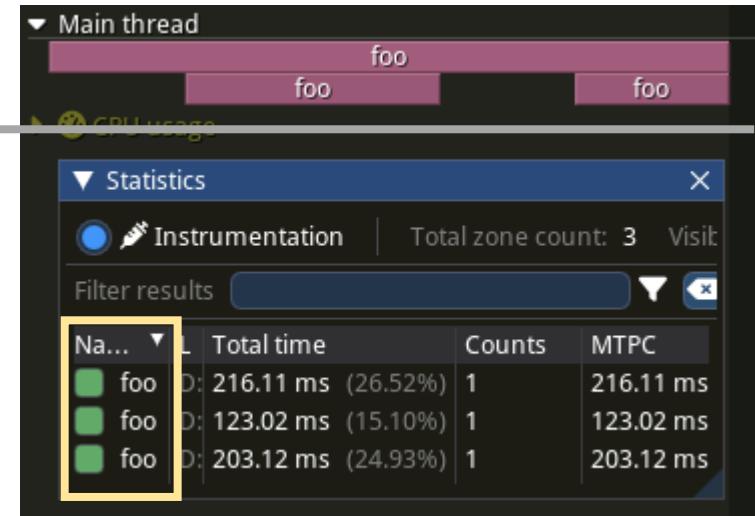
Source: <https://www.mobygames.com/game/657/simcity-2000/promo/group-17786/image-185051/>

Tracy Zoning 101

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

thread [timeline](#)

zone [statistics](#)

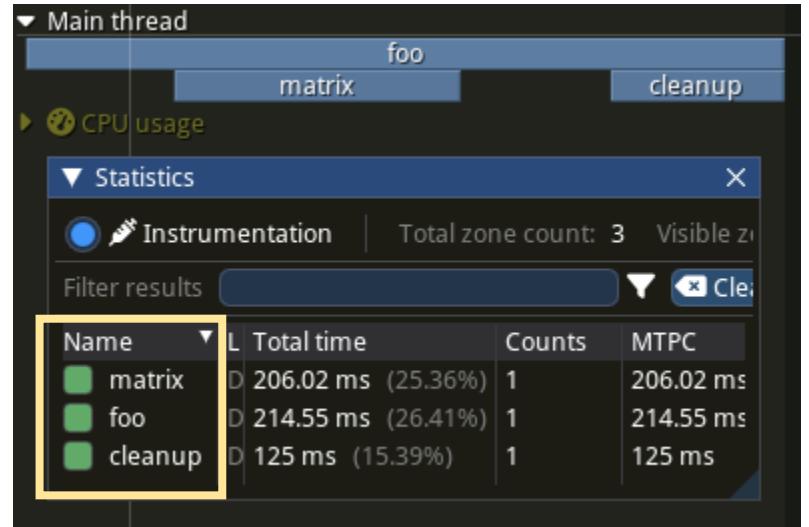


- **ZoneScoped**

- 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");
        ...
    }
}
```



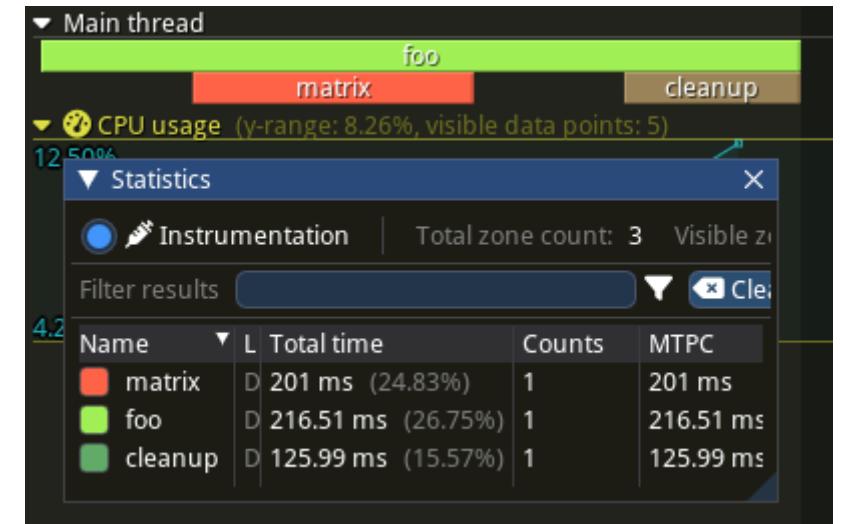
- **ZoneScopedN**
 - only 1 zone per scope
 - zone color is automatic (per-thread and depth)
 - **you name it!** (must be *persistent string literal*)

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");  
        ...  
    }
}
```

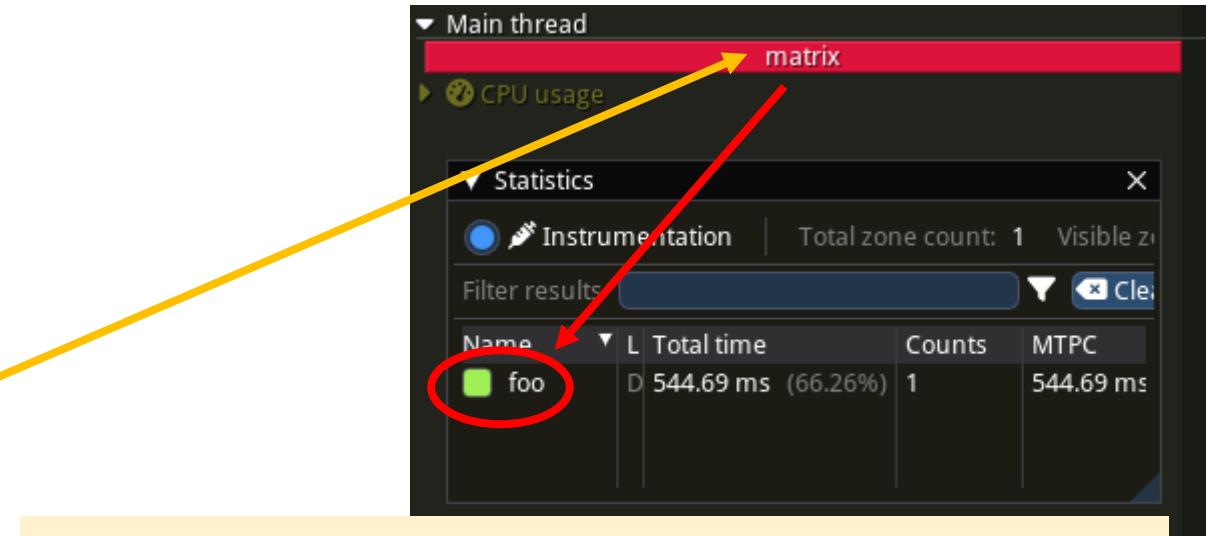


- **ZoneScoped[N]C**
 - only 1 zone per scope
 - you [may] name it
 - **you pick the color**
 - hexadecimal RGB triplet: 0xRRGGBB
 - or [X11 color names](#) (through **tracy::Color**)



Tracy Zoning 201

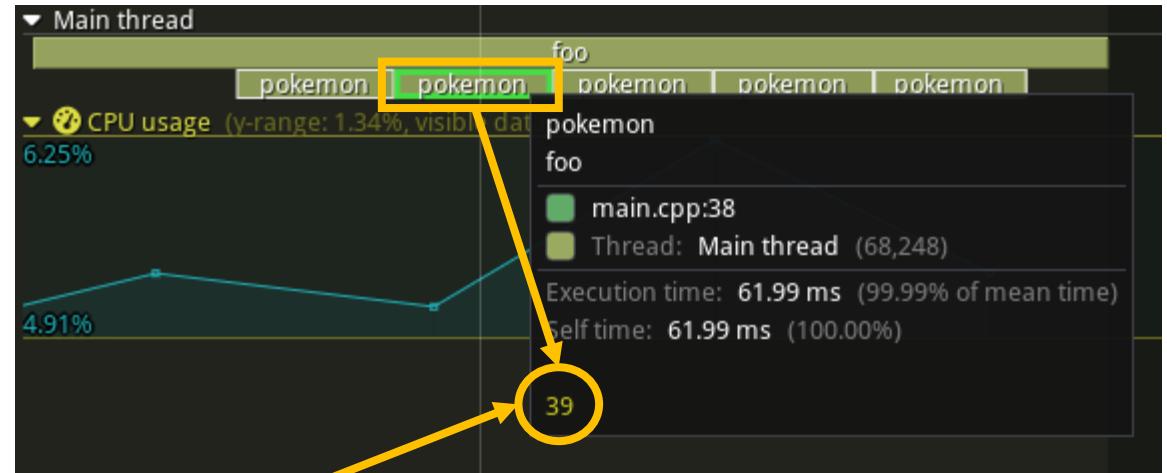
```
int foo()
{
    ZoneScoped;
    ...
    if (!spoon) {
        ZoneName("matrix", 6);
        ...
    }
    ...
    if (error != 0) {
        ZoneColor(Crimson);
        ...
    }
}
```



- **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

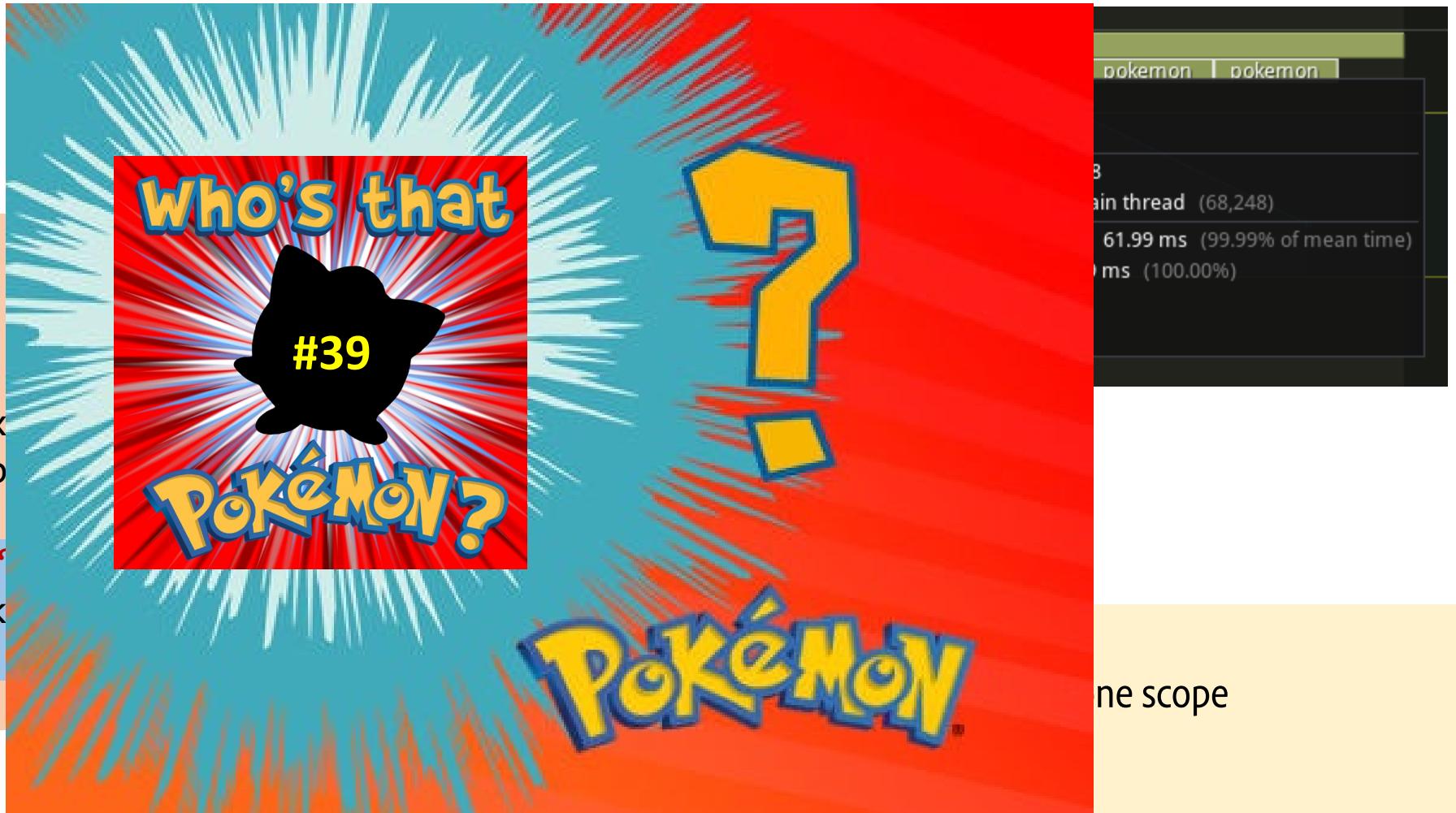
```
int foo()
{
    ZoneScoped;
    ...
    Pokedex pokedex;
    ...
    ZoneValue(pokedex.size());
    for (auto& pokemon : pokedex)
    {
        ZoneScopedN("pokemon");
        ZoneValue(pokemon.id);
        ...
    }
}
```



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

Tracy Zoning 201

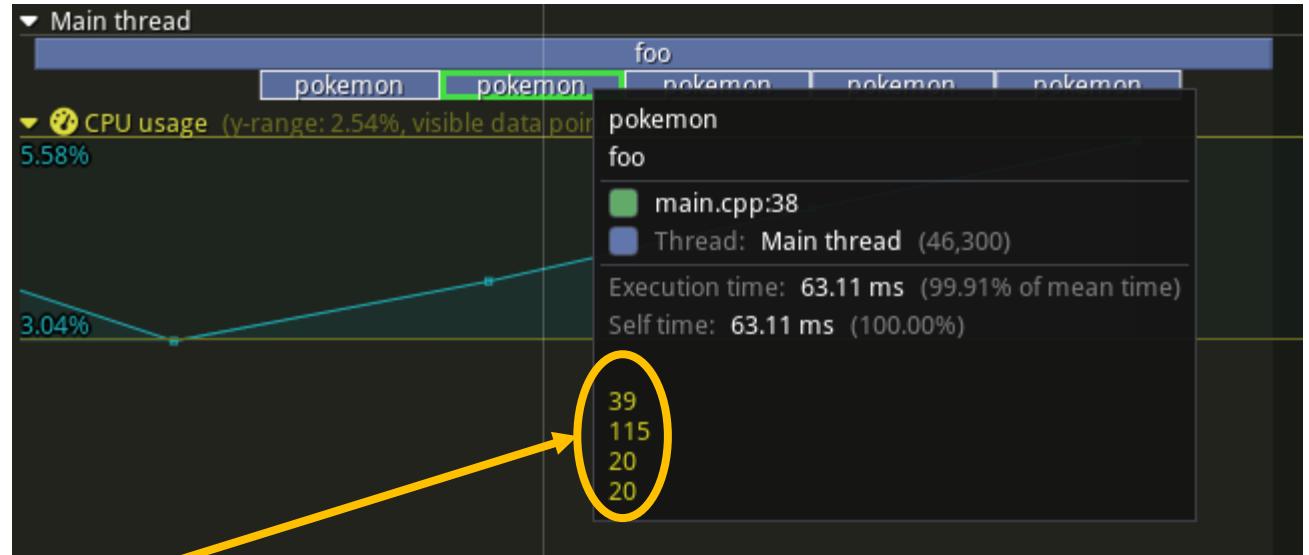
```
int foo()
{
    ZoneScoped;
    ...
    Pokedex pokedex;
    ...
    ZoneValue(pokedex)
    for (auto& pokemo
    {
        ZoneScopedN("Pokedex")
        ZoneValue(pok
        ...
    }
}
```



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);
        ...
    }
}
```

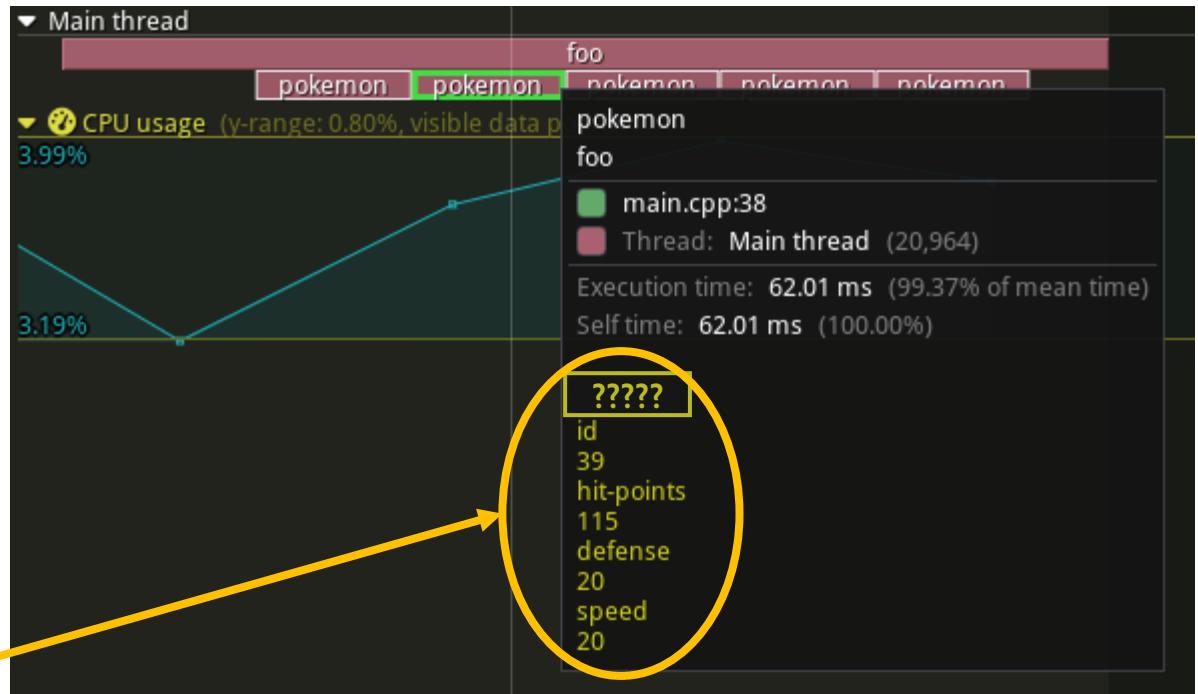


- **ZoneValue**

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

Tracy Zoning 201

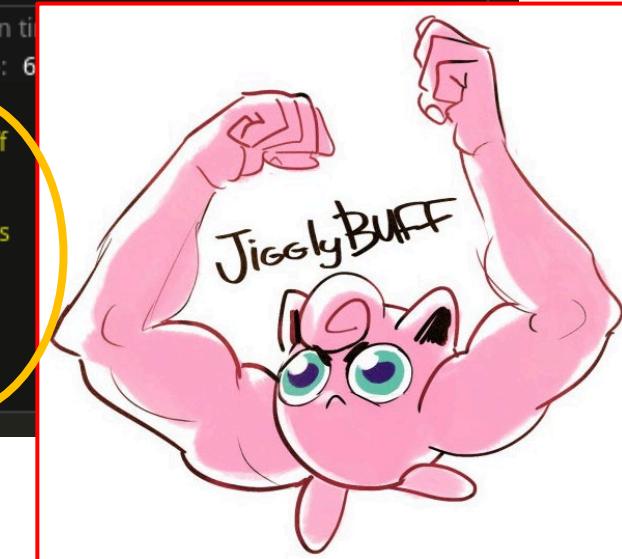
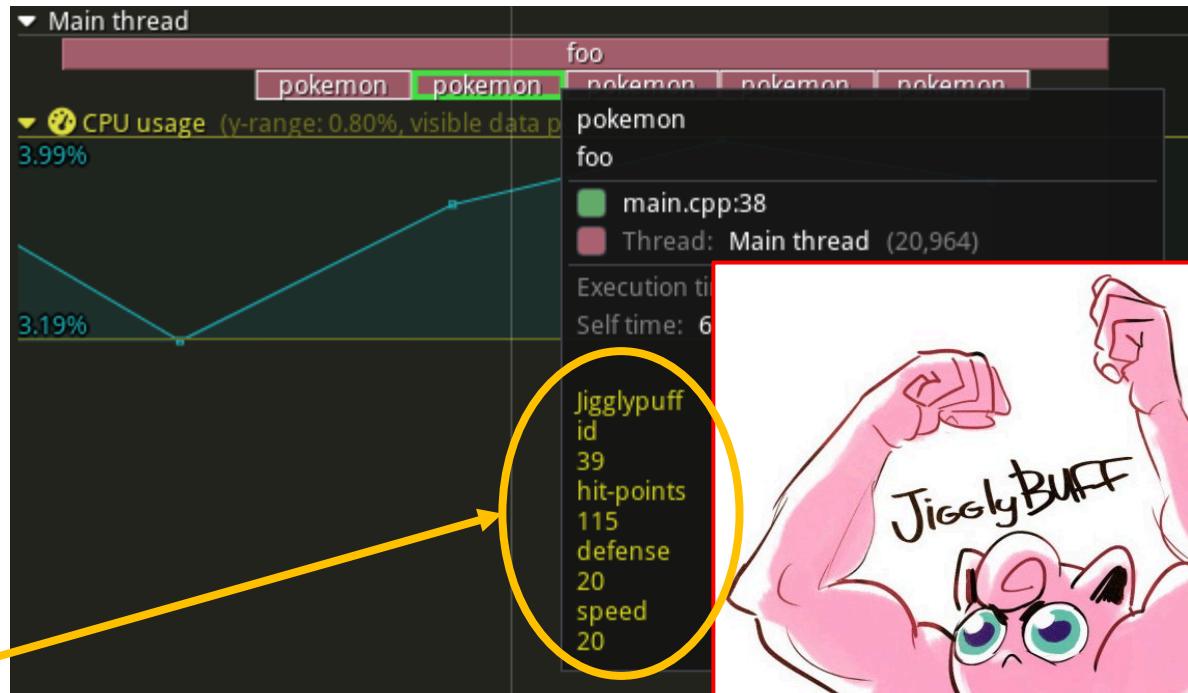
```
int foo()
{
    ZoneScoped;
    Pokedex pokedex;
    for (auto& pokemon : pokedex)
    {
        ZoneScopedN("pokemon");
        ...
        ZoneText("hit-points", 10);
        ZoneValue(pokemon.hp);
        ZoneText("defense", 7);
        ZoneValue(pokemon.def);
        ...
    }
}
```



- **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);
        ...
    }
}
```



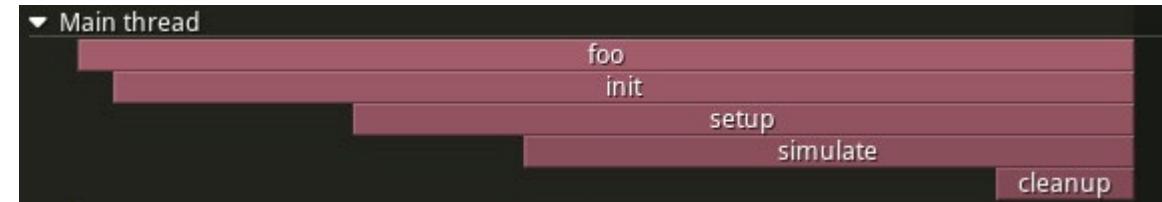
Source: <https://www.reddit.com/r/pokemon/comments/3o5je7/jigglybuff/>

- **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** (on/off) a zone **programmatically**

Tracy Zoning 301

ZoneScoped[N|C|S]

ZoneColor(...)

ZoneValue(...)

ZoneText(...)

ZoneName(...)

ZoneIsActive()

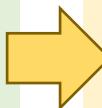
ZoneNamed[N|C|S]

ZoneColorV(varname, ...)

ZoneValueV(varname, ...)

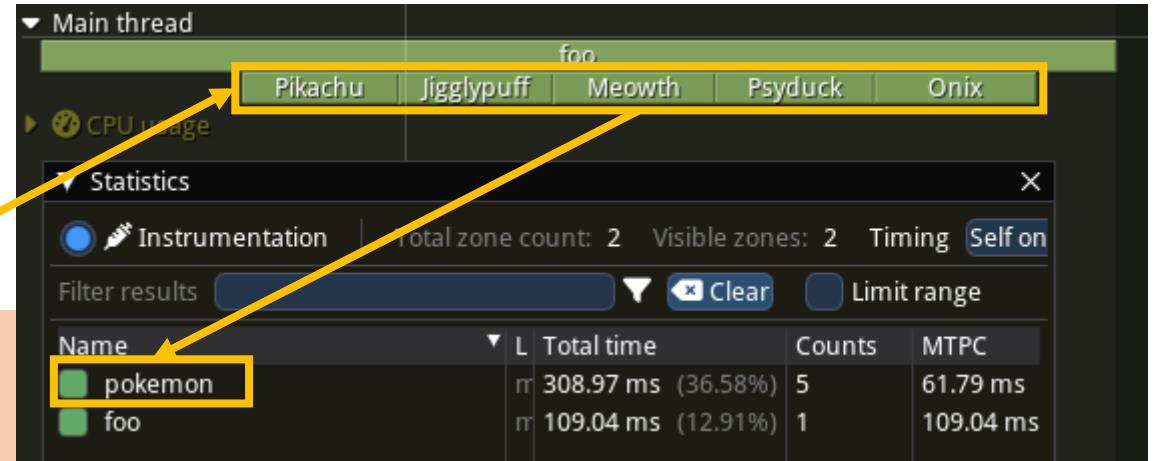
ZoneTextV(varname, ...)

ZoneNameV(varname, ...)



Tracy Zoning 401

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



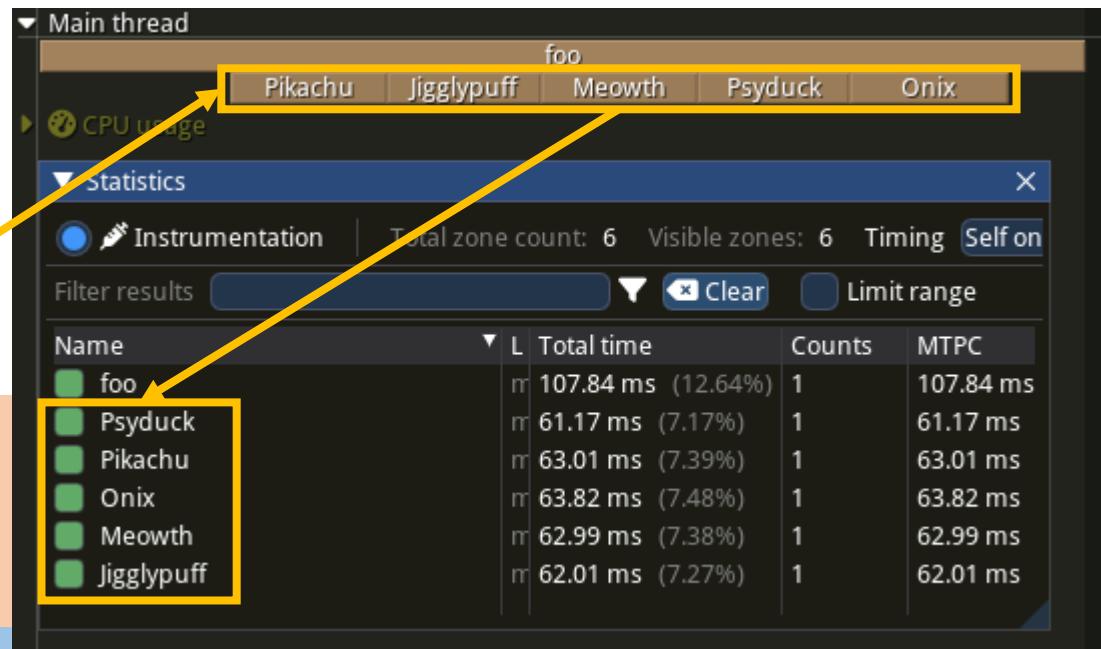
Tracy Zoning 401

```
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

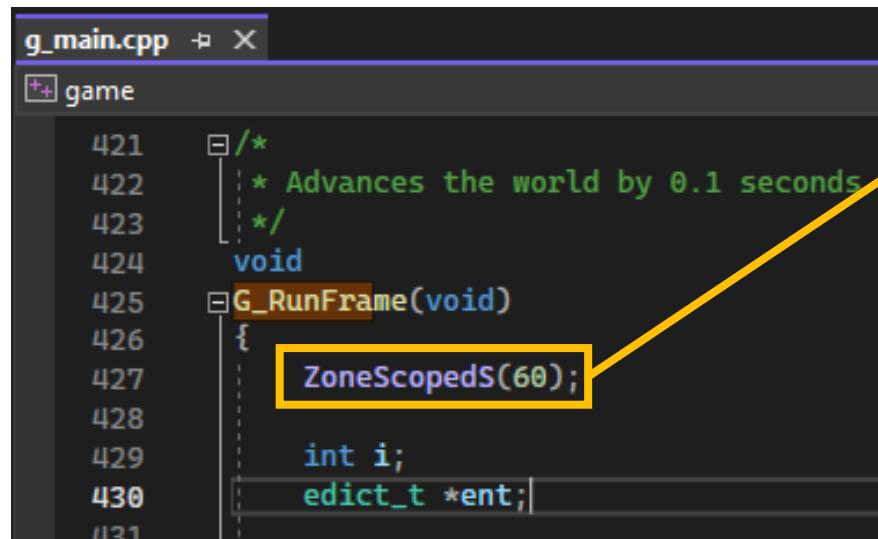
In order to prevent problems caused by unloadable code, described in section 3.1.1, transient zones copy the source location data to an on-heap buffer. This way, the requirement on the string literal data being accessible for the rest of the program lifetime is relaxed, at the cost of increased memory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

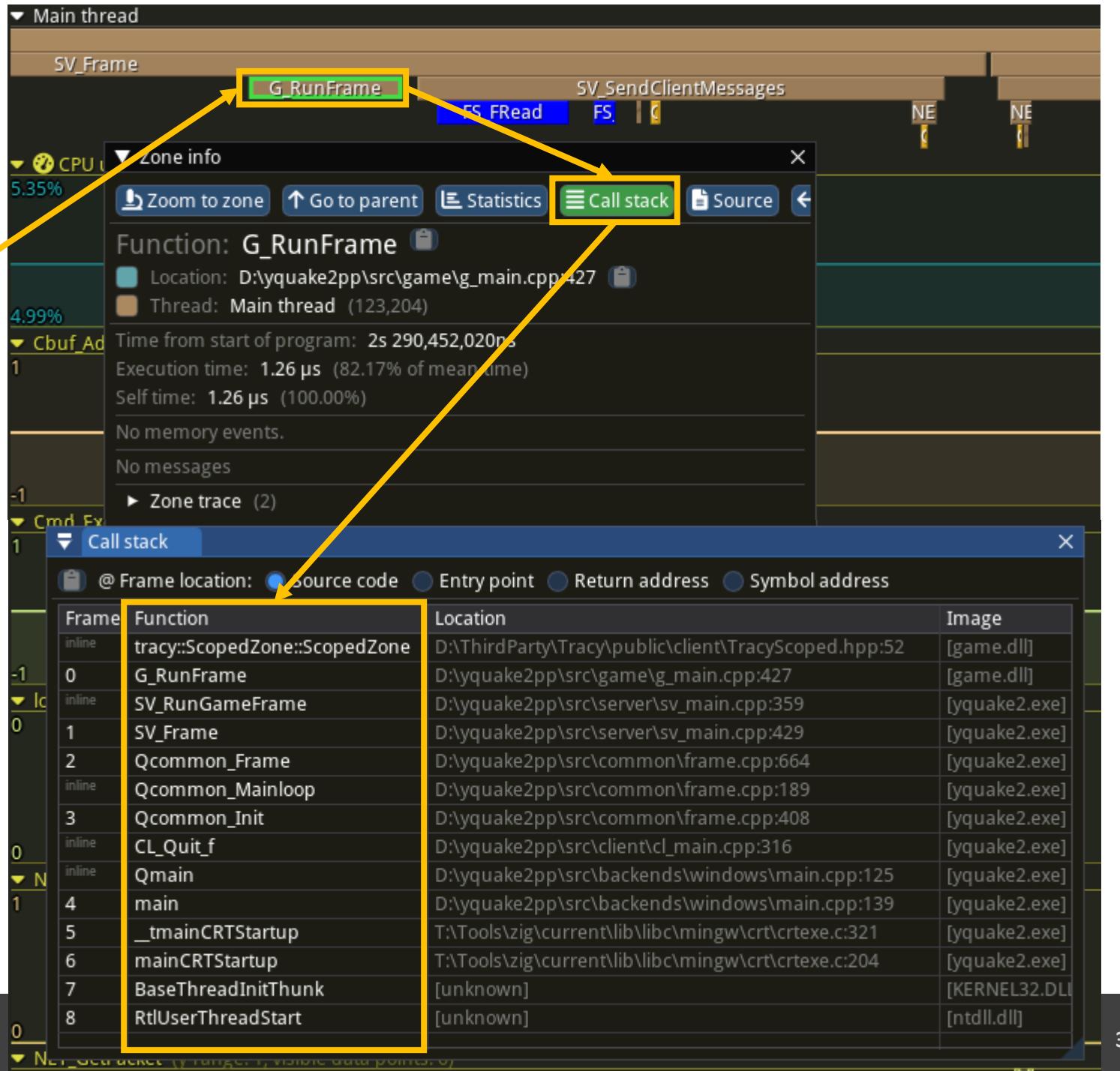
Tracy Zoning 501



```
g_main.cpp  X
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

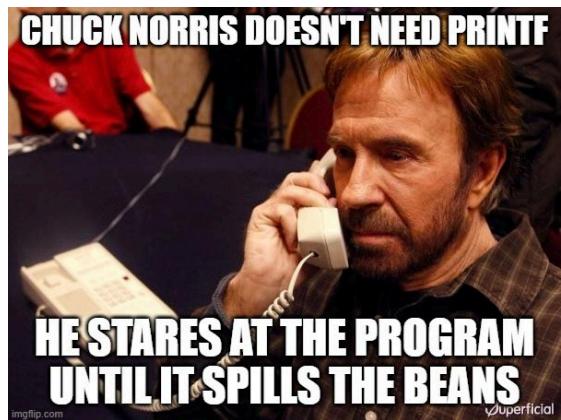
Recap (cheat sheet)

scope delimiters		for the lowest overhead possible		
Zone	Scoped	[N]	[C]	[S]
	Transient	[N]		[S]
	Color			
	Name			
	Value	[V]		
scope attributes				

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!

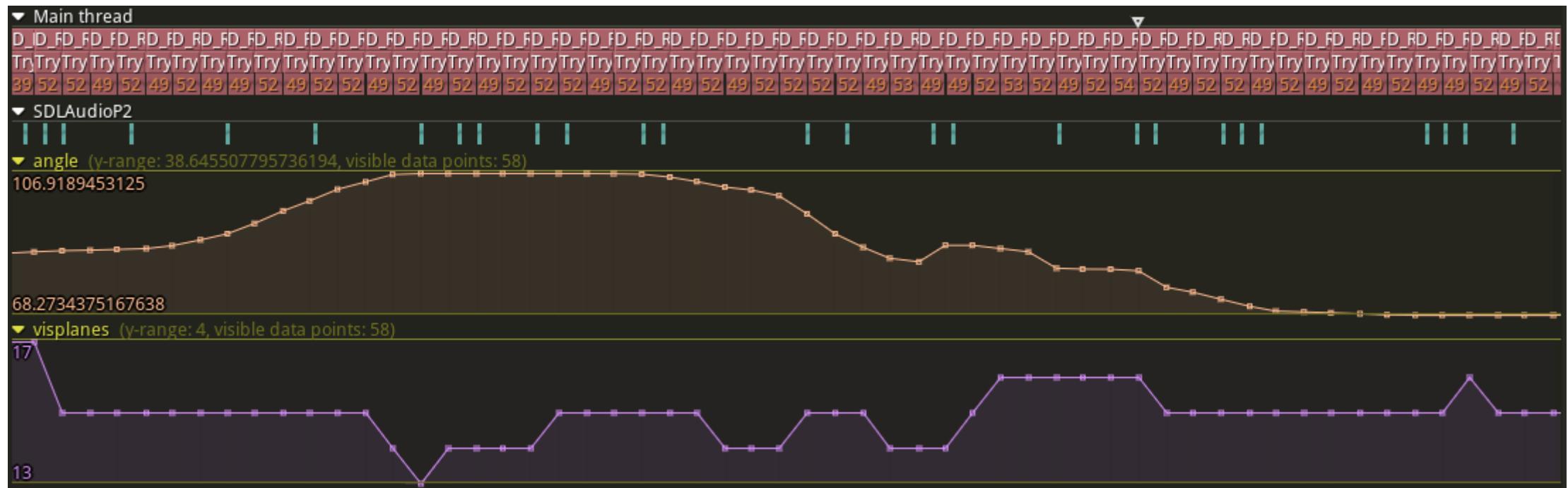
The screenshot shows the Tracy debugger interface. The top pane, "Main thread", displays a timeline with several events: "Swap BuU", "Swap Buffers", "Stop Lantern", "Start Lantern", and "GUI Update". Below this is the "Messages" pane, which is highlighted with a blue border. The "Messages" pane has a search bar with "Filter messages" and a "Clear" button. It also shows "Total message count: 159" and "Visible messages: 159". The list of messages is organized by time and thread. A yellow arrow points from the "persistent string literal" text in the Chuck Norris meme towards the "Messages" pane.

Time	Thread	Message
3s 850,505,210ns	Lantern Renderer (GPU) (58,292)	ecclair (timing): Found 400 light-emitting triangle(s)
3s 909,796,918ns	Lantern Renderer (GPU) (58,292)	ecclair: Creating bottom-level acceleration structure
3s 913,235,212ns	Lantern Renderer (GPU) (58,292)	ecclair (timing): Created 8 bottom-level acceleration
3s 913,677,780ns	Lantern Renderer (GPU) (58,292)	ecclair: Compacting 8 bottom-level accelera
3s 915,435,222ns	Lantern Renderer (GPU) (58,292)	ecclair (timing): Compacted 8 bottom-level accelera
3s 915,834,133ns	Lantern Renderer (GPU) (58,292)	ecclair: Creating top-level acceleration structure...
3s 921,000,822ns	Lantern Renderer (GPU) (58,292)	ecclair: Scene:
3s 921,393,717ns	Lantern Renderer (GPU) (58,292)	ecclair: Bounding box: [-57.46, 6.0180078] x [0, 37
3s 921,999,181ns	Lantern Renderer (GPU) (58,292)	ecclair: Center: (-25.720995, 18.991982, 0)
3s 922,576,876ns	Lantern Renderer (GPU) (58,292)	ecclair: Diameter: 108.95981
3s 925,504,940ns	Lantern Renderer (GPU) (58,292)	ecclair: Creating bottom-level acceleration structure
3s 926,510,549ns	Lantern Renderer (GPU) (58,292)	ecclair (timing): Created 0 bottom-level acceleration
3s 927,105,899ns	Lantern Renderer (GPU) (58,292)	ecclair: Creating top-level acceleration structure...
3s 928,081,567ns	Lantern Renderer (GPU) (58,292)	ecclair (timing): Found 400 light-emitting triangle(s)
3s 931,773,179ns	Lantern Renderer (GPU) (58,292)	ecclair_vk_framebuffer: Creating persistent pixels, k
3s 933,207,281ns	Lantern Renderer (GPU) (58,292)	ecclair_vk_framebuffer: Creating persistent pixels, k
6s 069,887,760ns	Main thread (78,860)	Lantern CPU Check: Warning: No AVX available, Fal
6s 071,344,285ns	Main thread (78,860)	Lantern CPU Check: Warning: No FP16 (HALF-PREC
6s 072,370,240ns	Main thread (78,860)	TBB engage
6s 086,996,305ns	tbb worker 1 (72,712)	TBB engage
6s 087,021,622ns	tbb worker 7 (78,812)	TBB engage
6s 088,598,012ns	Main thread (78,860)	reading runtime_texture_binary_header_v4015
6s 106,293,508ns	Main thread (78,860)	reading runtime_texture_binary_header_v4015
6s 124,248,815ns	Main thread (78,860)	reading runtime_texture_binary_header_v4015
6s 141,332,404ns	tbb worker 29 (84,036)	TBB engage
6s 141,335,770ns	tbb worker 24 (58,764)	TBB engage
6s 141,336,011ns	tbb worker 57 (60,140)	TBB engage

Plots

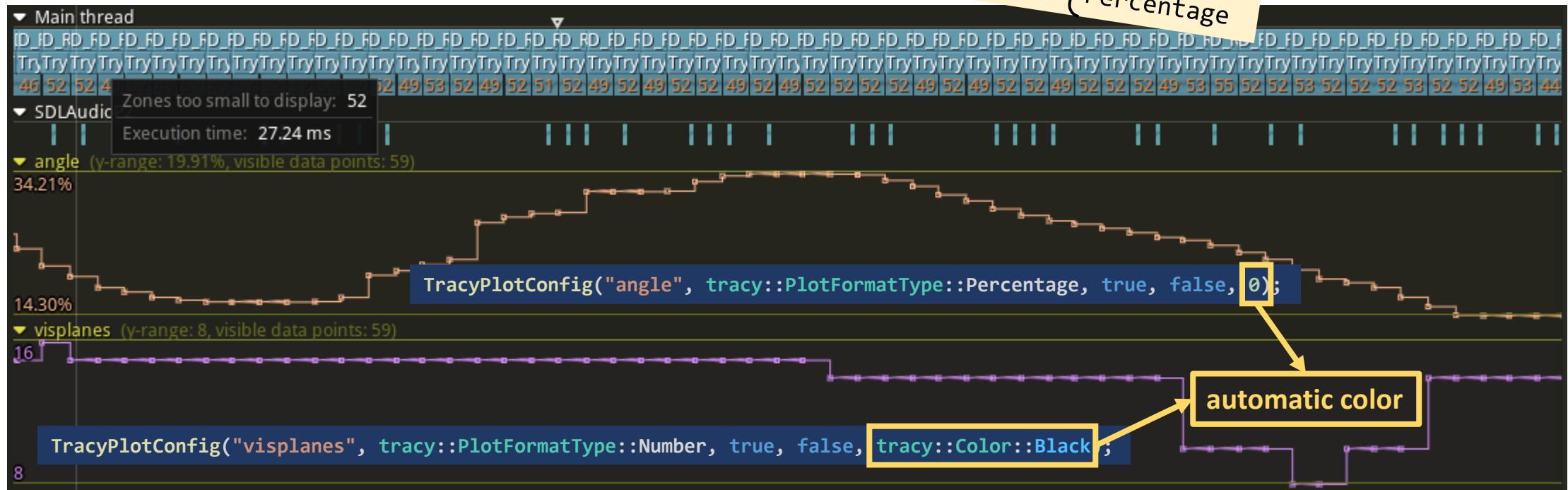
- `TracyPlot(name, value)`
persistent int64, float
string literal or double

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



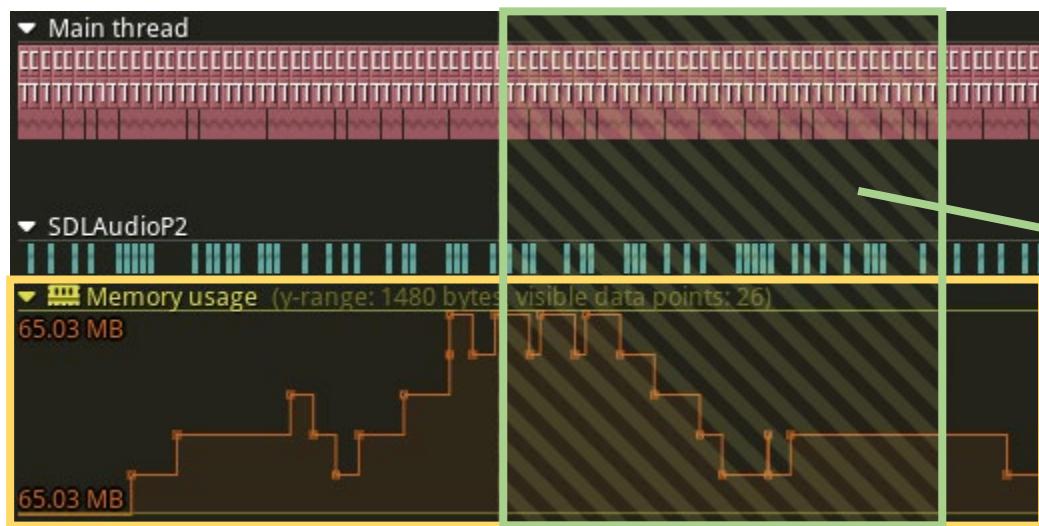
Plots

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



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

Memory																																																																																																																																															
Memory pool:		Default allocator																																																																																																																																													
Total allocations: 18,472		Active allocations: 4,435		Memory usage: 65.03 MB		Memory span:																																																																																																																																									
Active allocations (4,435) Memory usage: 65.03 MB		Address	Size	Appeare...	Duration	Thread	Zone alloc	Zone free																																																																																																																																							
<table border="1"><thead><tr><th>Address</th><th>Size</th><th>Appeare...</th><th>Duration</th><th>Thread</th><th>Zone alloc</th><th>Zone free</th><th> </th><th> </th></tr></thead><tbody><tr><td>0x2657fa3d5f</td><td>6 bytes</td><td>275,887,006ns</td><td>22.66 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2657fa56af</td><td>24 bytes</td><td>275,918,470ns</td><td>22.66 s</td><td>Main thread</td><td>doom_malloc</td><td>active</td><td></td><td></td></tr><tr><td>0x2650c0f504</td><td>32 MB</td><td>3s 756,820,177</td><td>19.18 s</td><td>Main thread</td><td>doom_malloc</td><td>active</td><td></td><td></td></tr><tr><td>0x2650b9a94z</td><td>0 bytes</td><td>3s 756,900,165</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650bf6819</td><td>11 bytes</td><td>3s 757,173,674</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650bf682f</td><td>15 bytes</td><td>3s 757,223,345</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650b9a957</td><td>2 bytes</td><td>3s 757,446,604</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650bf6855</td><td>23 bytes</td><td>3s 757,454,096</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650b9a925</td><td>7 bytes</td><td>3s 757,455,801</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650be6dea</td><td>25 bytes</td><td>3s 757,458,045</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650b9a920</td><td>5 bytes</td><td>3s 757,459,964</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650bf6807</td><td>9 bytes</td><td>3s 757,461,717</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650bf6831</td><td>21 bytes</td><td>3s 757,463,483</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr><tr><td>0x2650bf6833</td><td>10 bytes</td><td>3s 757,465,088</td><td>19.18 s</td><td>Main thread</td><td>doom_strdup</td><td>active</td><td></td><td></td></tr></tbody></table>									Address	Size	Appeare...	Duration	Thread	Zone alloc	Zone free			0x2657fa3d5f	6 bytes	275,887,006ns	22.66 s	Main thread	doom_strdup	active			0x2657fa56af	24 bytes	275,918,470ns	22.66 s	Main thread	doom_malloc	active			0x2650c0f504	32 MB	3s 756,820,177	19.18 s	Main thread	doom_malloc	active			0x2650b9a94z	0 bytes	3s 756,900,165	19.18 s	Main thread	doom_strdup	active			0x2650bf6819	11 bytes	3s 757,173,674	19.18 s	Main thread	doom_strdup	active			0x2650bf682f	15 bytes	3s 757,223,345	19.18 s	Main thread	doom_strdup	active			0x2650b9a957	2 bytes	3s 757,446,604	19.18 s	Main thread	doom_strdup	active			0x2650bf6855	23 bytes	3s 757,454,096	19.18 s	Main thread	doom_strdup	active			0x2650b9a925	7 bytes	3s 757,455,801	19.18 s	Main thread	doom_strdup	active			0x2650be6dea	25 bytes	3s 757,458,045	19.18 s	Main thread	doom_strdup	active			0x2650b9a920	5 bytes	3s 757,459,964	19.18 s	Main thread	doom_strdup	active			0x2650bf6807	9 bytes	3s 757,461,717	19.18 s	Main thread	doom_strdup	active			0x2650bf6831	21 bytes	3s 757,463,483	19.18 s	Main thread	doom_strdup	active			0x2650bf6833	10 bytes	3s 757,465,088	19.18 s	Main thread	doom_strdup	active		
Address	Size	Appeare...	Duration	Thread	Zone alloc	Zone free																																																																																																																																									
0x2657fa3d5f	6 bytes	275,887,006ns	22.66 s	Main thread	doom_strdup	active																																																																																																																																									
0x2657fa56af	24 bytes	275,918,470ns	22.66 s	Main thread	doom_malloc	active																																																																																																																																									
0x2650c0f504	32 MB	3s 756,820,177	19.18 s	Main thread	doom_malloc	active																																																																																																																																									
0x2650b9a94z	0 bytes	3s 756,900,165	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650bf6819	11 bytes	3s 757,173,674	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650bf682f	15 bytes	3s 757,223,345	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650b9a957	2 bytes	3s 757,446,604	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650bf6855	23 bytes	3s 757,454,096	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650b9a925	7 bytes	3s 757,455,801	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650be6dea	25 bytes	3s 757,458,045	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650b9a920	5 bytes	3s 757,459,964	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650bf6807	9 bytes	3s 757,461,717	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650bf6831	21 bytes	3s 757,463,483	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
0x2650bf6833	10 bytes	3s 757,465,088	19.18 s	Main thread	doom_strdup	active																																																																																																																																									
> Memory map																																																																																																																																															
> Bottom-up call stack tree																																																																																																																																															
> Top-down call stack tree																																																																																																																																															

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 readers-writer mutex (`std::shared_mutex`)

What's Next?

Tips & Tricks
(& Gotchas)

Study Cases
from Adobe

Explore
Together

&

Time for
Questions

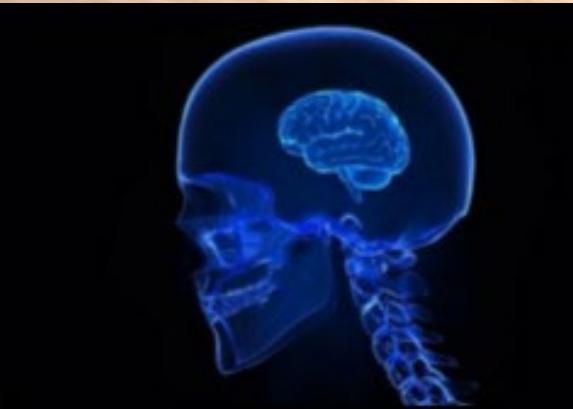
GPU
instrumentation
(very briefly)

A Look at Tracy's
Sampling Mode

Tips & Tricks



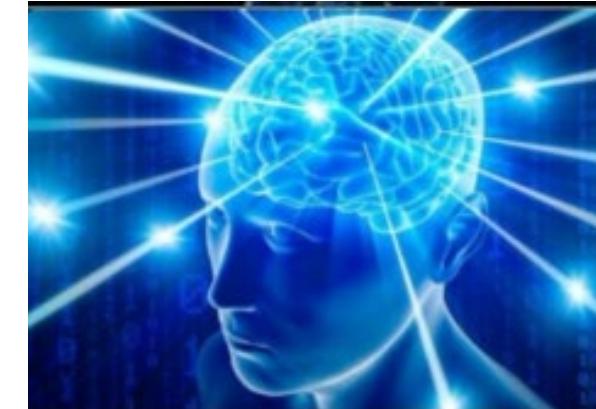
Apple users...



...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
- Waits
- Sleep
- Locks
- Memory
- Explicit
- Big-O
- I/O activity
- Unpredicted events

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)

good thing!

future.get()

ion code!

lockable

() et al.

compilation

, searching, container manipulations, ...

file, network, logging, ...

interrupts, user input activity, spurious wake-ups, ...

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!)



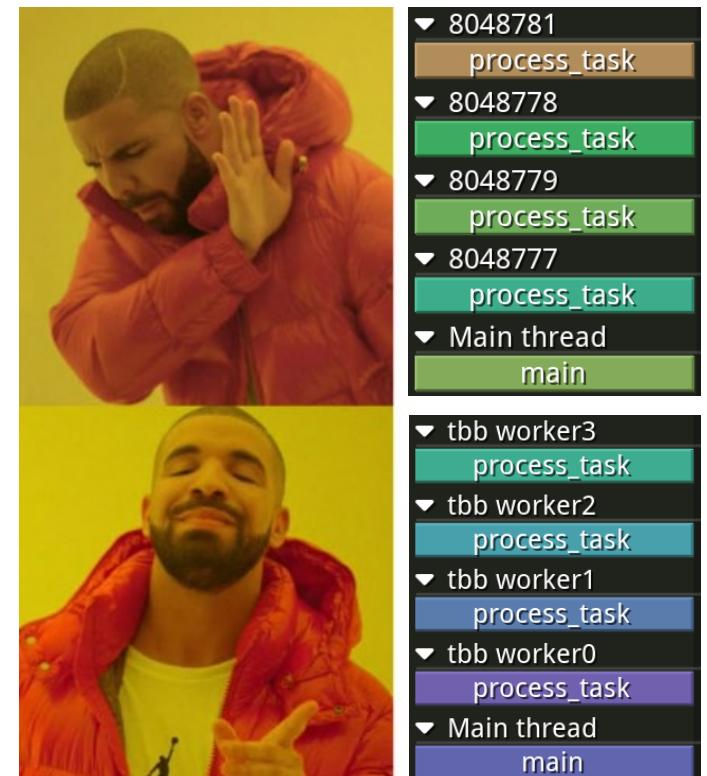
Tips & Tricks

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)

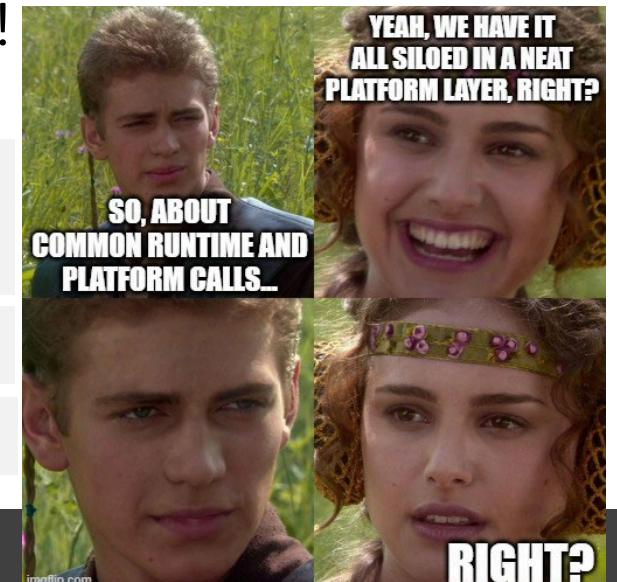
[LD PRELOAD](#)

[DYLD INSERT LIBRARIES](#)

[Microsoft Detours](#)

[weak symbol linkage interposition](#)

[MSVC linker /alternatename](#)



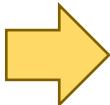
Tips & Tricks

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

Need to **instrument all overrides**...

Consider using **static delegate** interface to call the **virtual** method
(if only we could manipulate/later vtable entries... C-style (or COM-style) dispatch tables?)

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

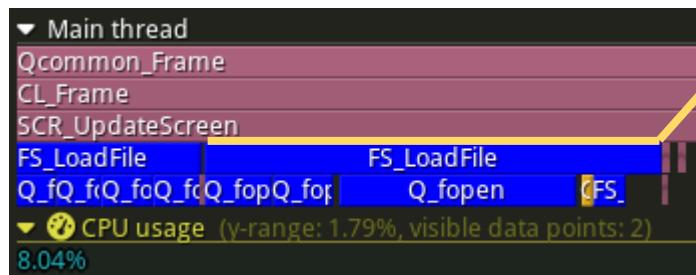


```
class Shape
{
public:
    float area() {
        ZoneScoped;
        return calculate_area();
    }
protected:
    virtual float calculate_area();
};
```

Tips & Tricks

Capture **callstack** on “terminal” zones for “code reconnaissance”

```
048  /*  
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  }
```



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]
2	R_LoadImage	D:\yquake2pp\src\client\refresh\soft\sw_image.cpp:593	[ref_soft.dll]
3	R_FindImage	D:\yquake2pp\src\client\refresh\soft\sw_image.cpp:697	[ref_soft.dll]
4	RE_Draw_PicScaled	D:\yquake2pp\src\client\refresh\soft\sw_draw.cpp:392	[ref_soft.dll]
5	M_Main_Draw	D:\yquake2pp\src\client\menu\menu.cpp:621	[yquake2.exe]
6	M_Draw	D:\yquake2pp\src\client\menu\menu.cpp:4973	[yquake2.exe]
7	SCR_UpdateScreen	D:\yquake2pp\src\client\cl_screen.cpp:1656	[yquake2.exe]
8	CL_Frame	D:\yquake2pp\src\client\cl_main.cpp:843	[yquake2.exe]
9	Qcommon_Frame	D:\yquake2pp\src\common\frame.cpp:680	[yquake2.exe]
10	Qcommon_Mainloop	D:\yquake2pp\src\common\frame.cpp:189	[yquake2.exe]
11	Qcommon_Init	D:\yquake2pp\src\common\frame.cpp:408	[yquake2.exe]
12	CL_Quit_f	D:\yquake2pp\src\client\cl_main.cpp:316	[yquake2.exe]
13	Qmain	D:\yquake2pp\src\backends\windows\main.cpp:125	[yquake2.exe]
14	main	D:\yquake2pp\src\backends\windows\main.cpp:139	[yquake2.exe]
15	_tmainCRTStartup	T:\Tools\zig\current\lib\libc\mingw\crt\crtexe.c:321	[yquake2.exe]
16	mainCRTStartup	T:\Tools\zig\current\lib\libc\mingw\crt\crtexe.c:204	[yquake2.exe]
17	BaseThreadInitThunk	[unknown]	[KERNEL32.DLL]
18	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()**

▼ Memory

Total allocations: 4,218 Active allocations: 1,058 Memory usage: 25.37 MB Memory span: 420.34 MB (?)

▼ @ Allocations

Enter memory address to search for Find Clear

▼ ❤ Active allocations (6) Memory usage: 5070 bytes

Address	Size	Appeared at	Duration	Thread	Zone alloc	Zone free	Call stack
0x15d1d16ebc	845 bytes	4s 200,359,212	1.02 s	Main thread	Qmalloc	Qfree	alloc [free]
0x15d1d6061f	845 bytes	4s 201,298,125	1.02 s	Main thread	Qmalloc	Qfree	alloc [free]
0x15d1d60655	845 bytes	4s 202,266,676	1.02 s	Main thread	Qmalloc	Qfree	alloc [free]
0x15d1d6068b	845 bytes	4s 203,251,339	1.02 s	Main thread	Qmalloc	Qfree	alloc [free]
0x15d1d57c50	845 bytes	4s 204,230,647	1.02 s	Main thread	Qmalloc	Qfree	alloc [free]
0x15d1d57c86	845 bytes	4s 205,185,405	1.02 s	Main thread	Qmalloc	Qfree	alloc [free]

▶ Memory map
▶ Bottom-up call stack tree
▶ Top-down call stack tree

For known object leaks, consider instrumenting constructors and destructors with TracyAlloc/Free

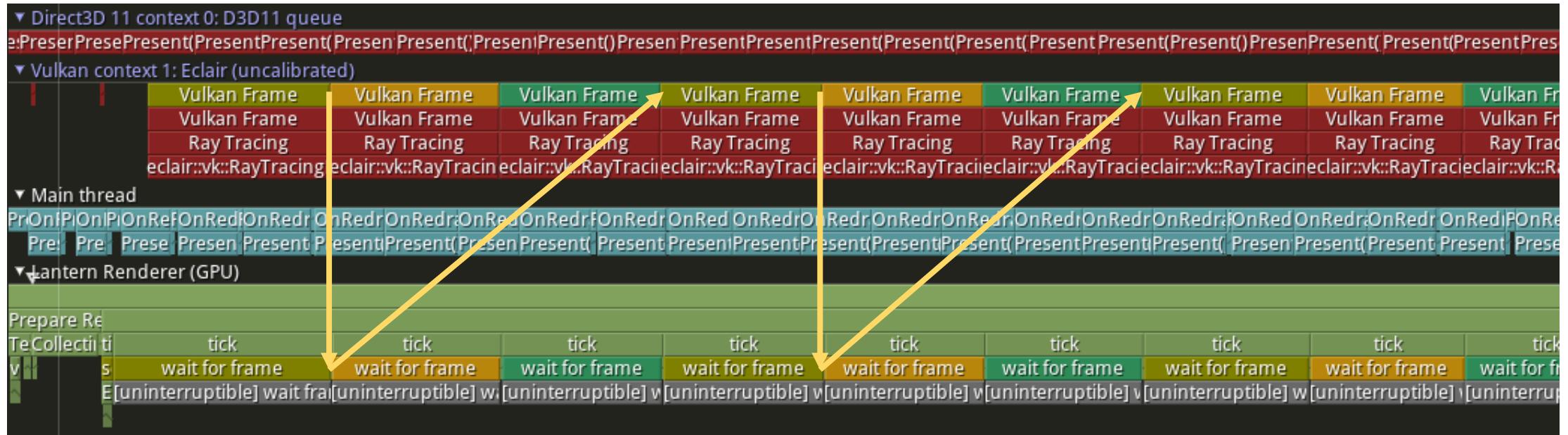
The call stack for the last allocation (0x15d1d57c86) is highlighted with a yellow box:

- 0. Qmalloc [ref_soft.dll]
- 1. R_LoadPic [ref_soft.dll]
▶ R_LoadImage [ref_soft.dll]
- 2. R_FindImage [ref_soft.dll]
- 3. RE_Draw_FindPic [ref_soft.dll]
- 4. M_DrawCursor [lyquake2.exe]
- 5. M_Main_Draw [lyquake2.exe]
- 6. M_Draw [lyquake2.exe]
- 7. SCR_UpdateScreen [lyquake2.exe]
- 8. CL_Frame [lyquake2.exe]
- 9. Qcommon_Frame [lyquake2.exe]
▶ Qcommon_Mainloop [lyquake2.exe]
- 10. Qcommon_Init [lyquake2.exe]
▶ CL_Quit_f [lyquake2.exe]
- 11. Qmain [lyquake2.exe]
- 12. main [lyquake2.exe]
- 13. __tmainCRTStartup [lyquake2.exe]
- 14. mainCRTStartup [lyquake2.exe]
- 15. BaseThreadInitThunk [KERNEL32.dll]
- 16. RtlUserThreadStart [ntdll.dll]



Tips & Tricks

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



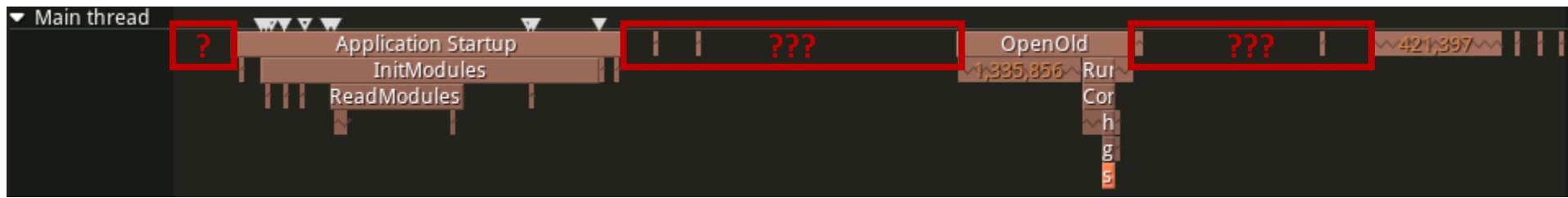
Tips & Tricks

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

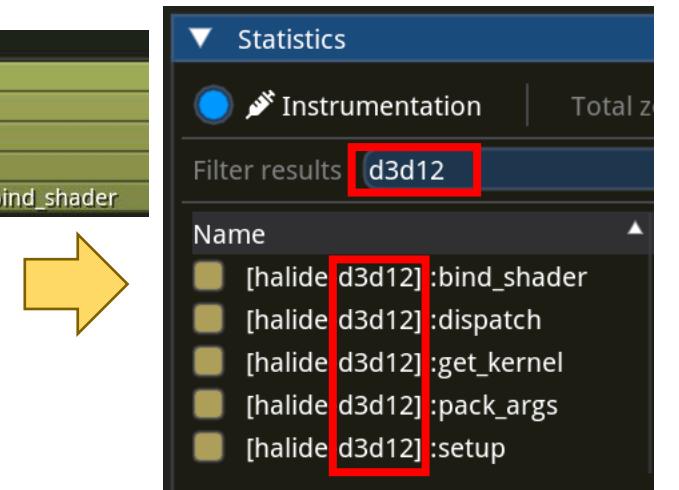
Add **API prefix** to zone name when no such prefix/namespace exists



Use the **:: separator !**

Tracy Profiler knows about it and will “split” on it for visualization

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

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

stairstep

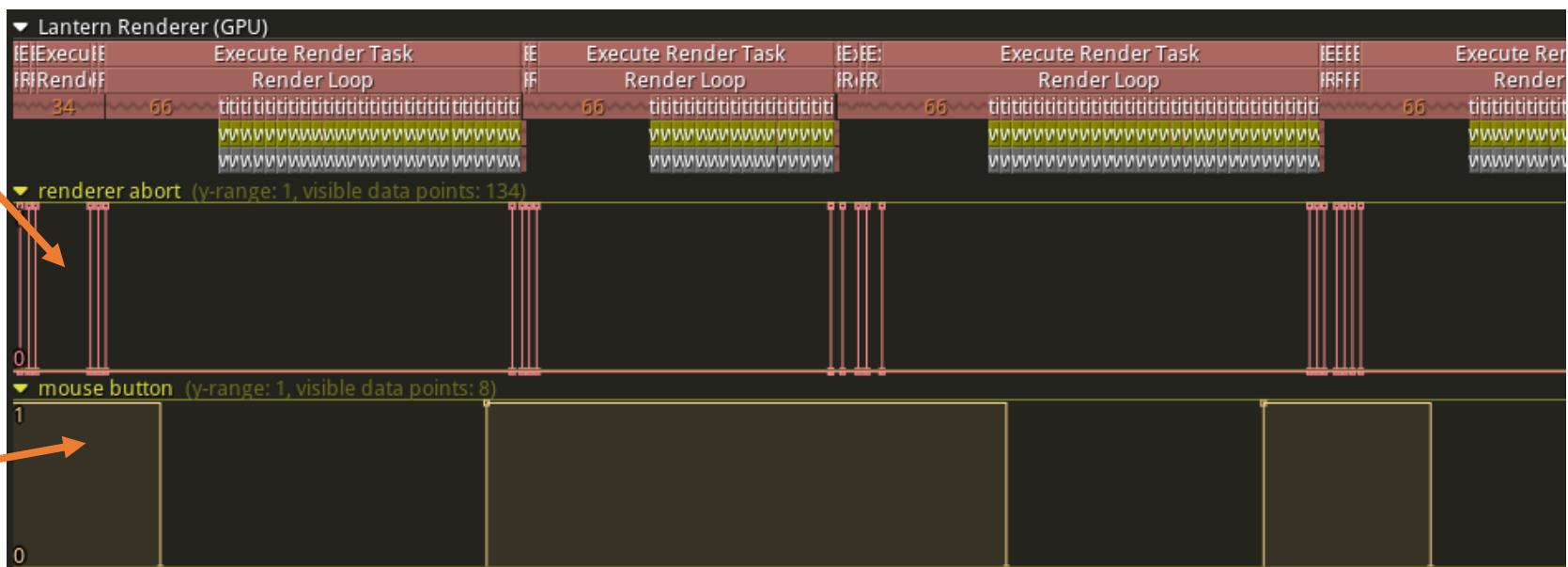
Diagram illustrating a pulse (heartbeat) plot. The timeline is marked by three dashed lines: 'up-tick', 'baseline', and 'down-tick'. Blue vertical bars represent events. The first bar starts at the baseline and ends at the up-tick. The second bar starts at the baseline and ends at the down-tick. The third bar starts at the up-tick and ends at the down-tick.

clock (sustain) plots

up-tick

baseline

down-tick



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!

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

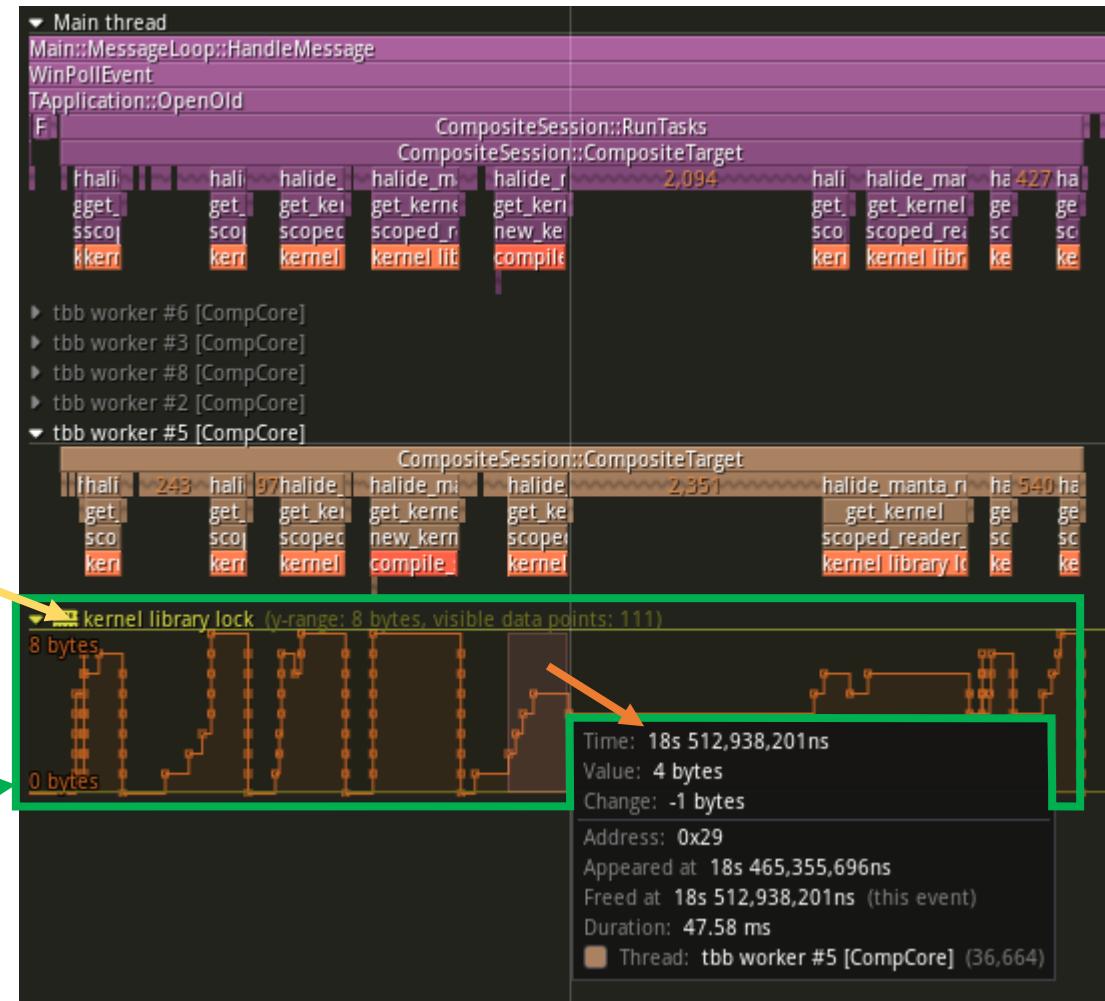
easily adaptable to reader-writer mutex as well
(std::shared_mutex and std::shared_lock)

Tips & Tricks

Measuring lock contention

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

[[nodiscard]]
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

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 staircase `TracyPlot` (uptick/downtick)
 - Assign unique `ZoneColor` to each task
- My **recommendation**: use `TracyAlloc` and `TracyFree`
(it’s basically a `stackable staircase plot!`)

Tips & Tricks

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

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

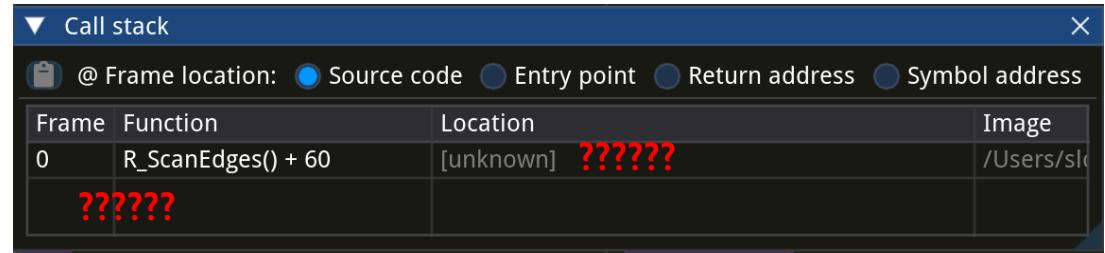
David Farrell's
pro-tip!
Md



Gotchas

Where are my Missing callstack symbols

- Compile with debug symbols
 - [gcc/clang] -g
 - [MSVC] /DEBUG
- DON'T omit frame pointers
 - [gcc/clang] -fno-omit-frame-pointer
 - [MSVC] not applicable for 64bit toolchain (/Oy- for 32bit toolchain)
- Make host executable symbols available to the dynamic linker
 - [MacOS] -export_dynamic
 - [Linux] -rdynamic or --export-dynamic (or -export_dynamic)
 - [Windows] not applicable (--export-dynamic is available in gcc/clang for WSL and MinGW, but no effect on PE executables, afaik)
- [MacOS] Make sure to call dsymutil on the executable *and* any dylib linked to it



A screenshot of a debugger's call stack window. The window title is "Call stack". At the top, there is a toolbar with a briefcase icon and the text "@ Frame location:". Below the toolbar are four radio buttons: "Source code" (selected), "Entry point", "Return address", and "Symbol address". The main area is a table with four columns: "Frame", "Function", "Location", and "Image". The first row shows frame 0, function R_ScanEdges() + 60, location [unknown], and image /Users/slc. The second row is partially visible, showing "???????" in all four columns.

Frame	Function	Location	Image
0	R_ScanEdges() + 60	[unknown] ??????	/Users/slc
	???????		

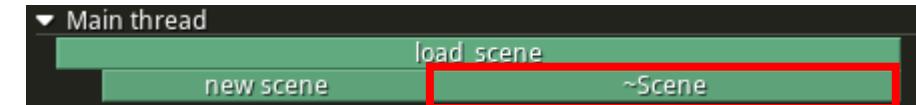
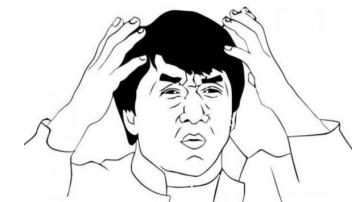
Refer to Tracy Manual: Section 3.11.1 – Debugging Symbols

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

```
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;
}
```

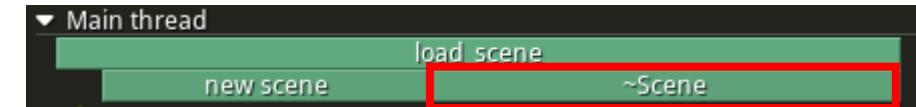
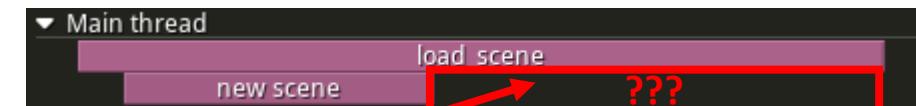


Gotchas

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

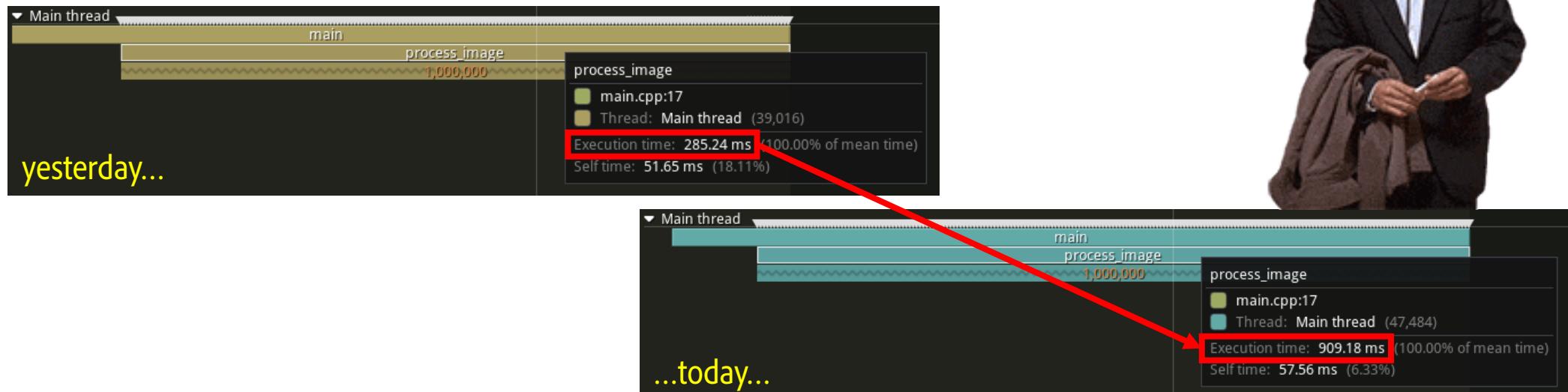
```
bool Viewport::load_scene(const char* filename)
{
    ZoneScoped;
    // ...
    std::shared_ptr<Scene> new_scene;
    {
        ZoneScopedN("new scene");
        new_scene = std::make_shared<Scene>(filename);
        if (!new_scene) return false;
    }
    m_scene = new_scene;
    return true;
}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



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!

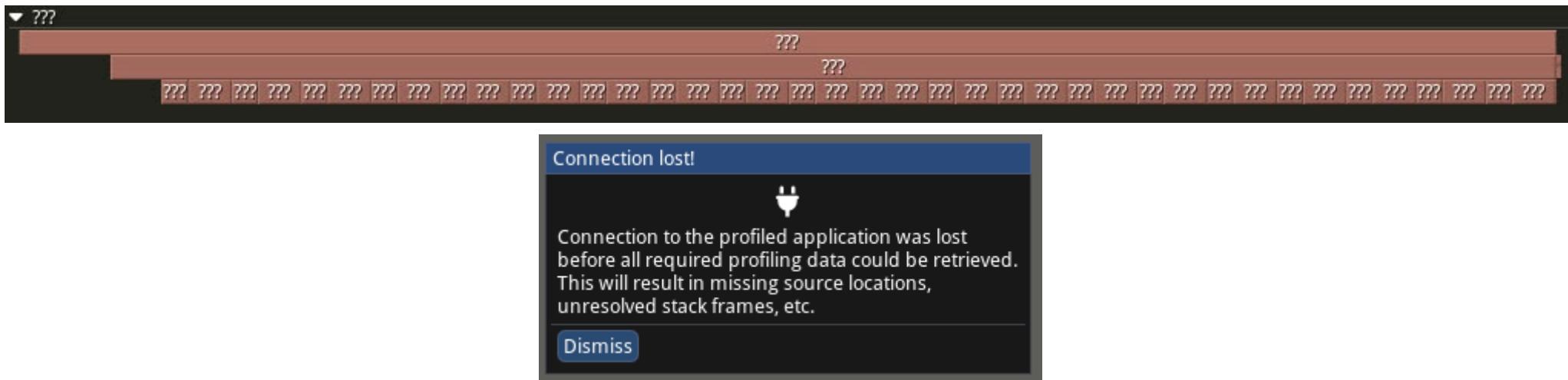
- hot **inner-loops**, small/inlined routines called frequently, ...
- **Profiling skew**: when profiling itself starts affecting performance
- May also affect **compiler optimizations**!

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

Gotchas

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

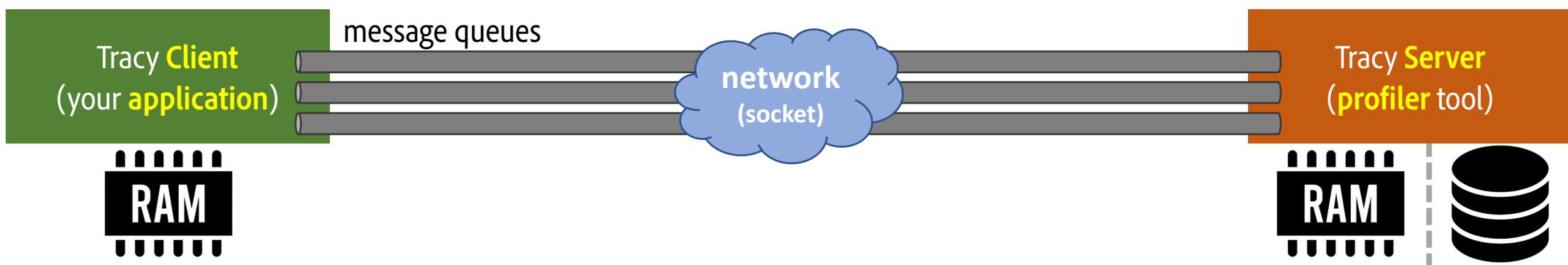
- just **wait** a little bit more (host application may not have sent this data yet)
- the host application may have **crashed** or terminated abruptly
- ensure zone name is a **persistent string**, or use **ZoneTransient()**
- if zone belongs to **plugin code**, make sure the plugin has **not been unloaded**



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**

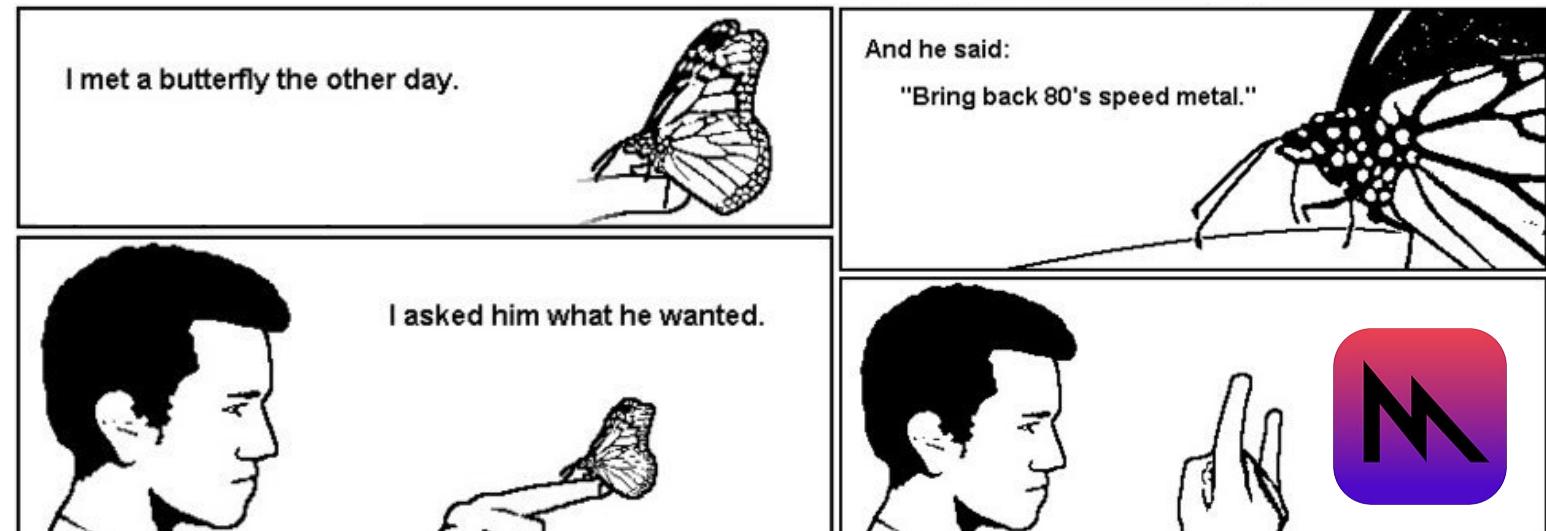


- Tip: #define `TRACY_ON_DEMAND` to delay producer until profiler gets connected
- Tip: use `capture` utility to “sink” Tracy events to `disk`
- Tip: consider running the profiler GUI on a **different machine**

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!
- 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 :-)



imgflip.com

Known Issues

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

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

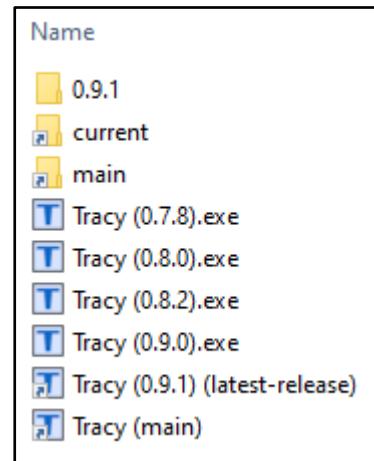
download pre-built binary (Windows-only):

<https://github.com/wolfpld/tracy/releases>

or build it yourself:

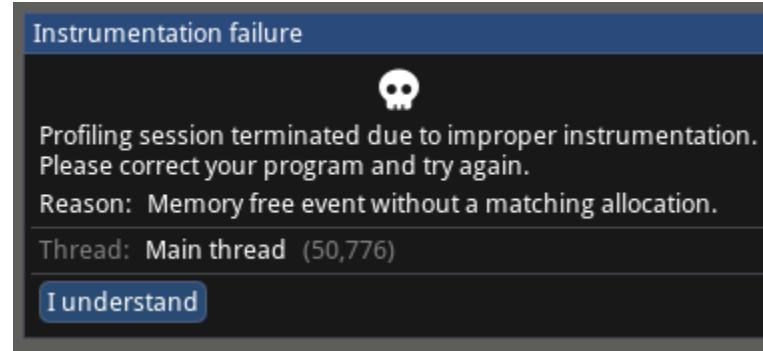
<tracy-repo>/update/build

- 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 locations¹⁶. 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

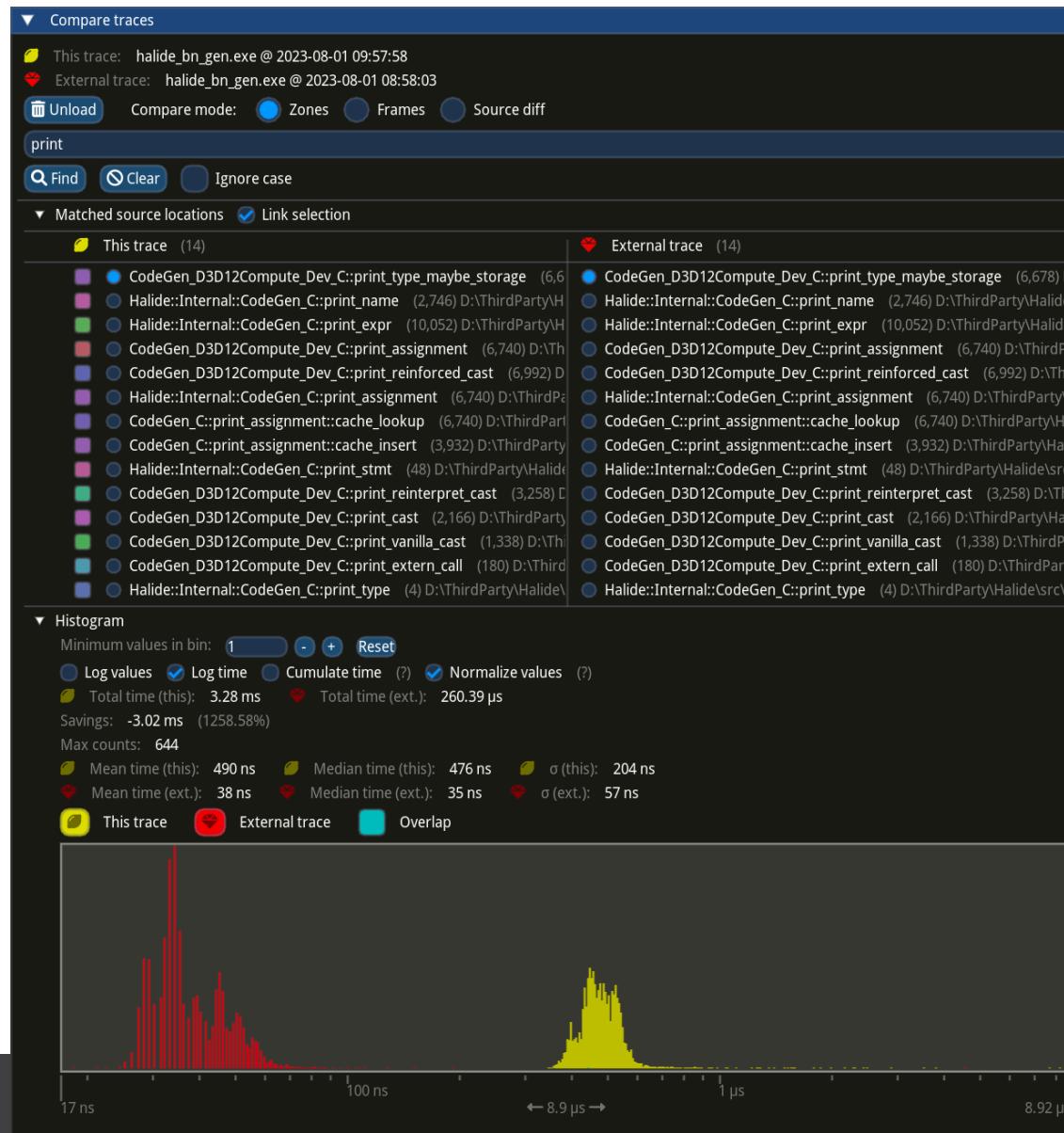
The following conditions also need to apply but don't trouble yourself with them too much. You would probably already know if you'd be breaking any.

- 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.

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...
- [Substance 3D Modeler] Improved parallel shader compilation
- [Substance 3D Stager] Eliminating GPU<->CPU copies with GPU API interop

Comparing Traces



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

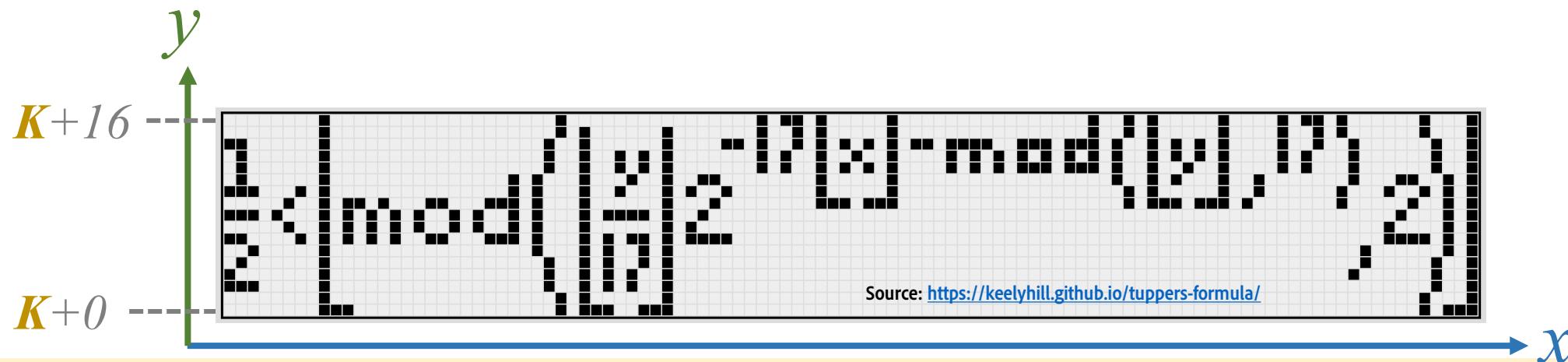
Manual instrumentation is **not necessary!**
(but you can have **both** coexist **together!**)



A Taste of Tracy's Sampling Mode

Tupper's Self-Referential Formula

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



K =

48584506361897134235820959624942020445814005879832445494830930850619347047088090284506447698655243648499972470249151191104116057391774078569197543265718
554420572104457358836818298237541396343382251994521916512843483329051311031153302413758765239264874613394906870130562295813219481113685339535565290850
0238750928568926945559742815463865107300491067230589335860525440966633126534936364395712556569593681518433485760526694016125126695142155053955451915378
5457525756590740540157929001765967965480064427829131488548259914721248506352686630476300

544 digits

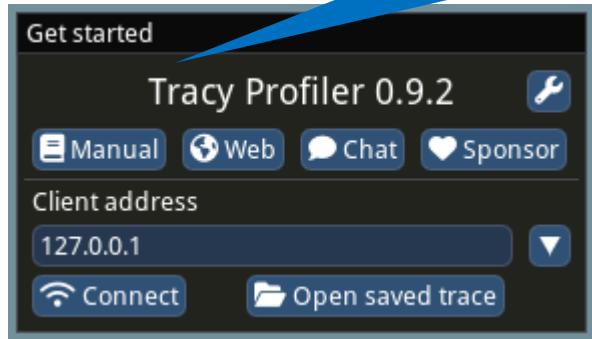
A New Speedrun Category!

Quake 2 Demo First Frame Any%

World Record: Marcos Slomp 1.32s

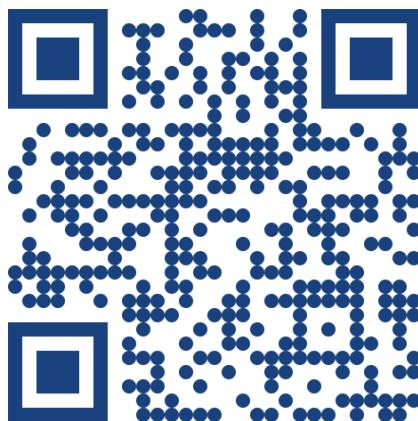
Help

all the help you need is in the **profiler window** already!



RTTM: Read the Tracy Manual (it's *REALLY* good!)
[tracy.pdf](#) (grab it from <https://github.com/wolfpld/tracy/releases>)

Discord server

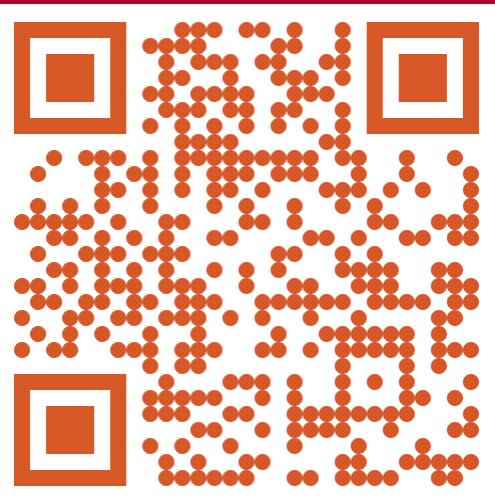


<https://discord.gg/pk78auc>

this->code

<https://github.com/slomp/cppcon/tracy>

C++ at Adobe!



developer.adobe.com/cpp

- Careers
- Events
- Training
- Blog Posts



research.adobe.com

- Careers
- Internships
 - Events
- Blog Posts

