UniKorn

Software Event Analyzer

Free and Open Source: Apache 2.0 License

https://github.com/michaelboth/EventAnalyzer

“Improving complex software can be like trying to find a *”*

Michael Both: Sep 1 2022

Unikorn

What's the Purpose?

Provides insight into a complex software environment over time, leading the way to significant optimizations.

Even seasoned experts are not immune to developing the dark

Tradition Ways of Trying to See Complexity in the Dark

Debuggers

- Well suited for tracking down a crash
- Painfully slow if stepping through breakpoints to understand long term behavior

Print Statements

- Well suited for periodic information
- Difficult to interpret print statements from concurrent streams
- Obtrusive to real-time applications

Function Profilers (e.g. GNU gprof, ARM MAP)

- Great to determine function averages across life of an app
- Difficult if trying to focus only on critical sections of an app
- Difficult if trying to look at a sub set of functions in a large pool of functions

Unikorn

Benefits of Unikorn

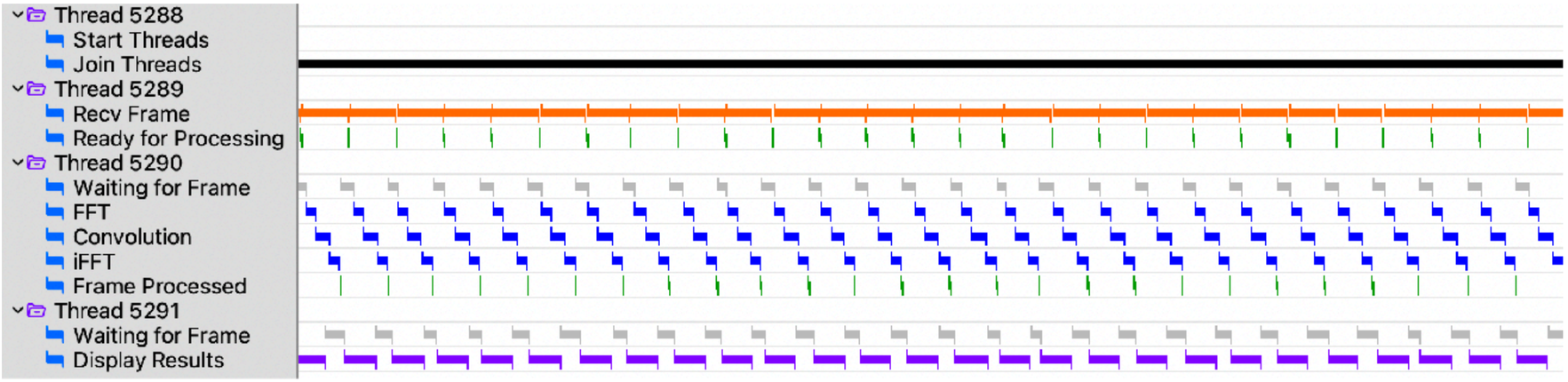
Goes well beyond debuggers, printf, and profilers

Using unikorn is like entering ‘The Matrix’ or ‘The Grid’.

The next few slides show a **TINY FRACTION** of what Unikorn can reveal.

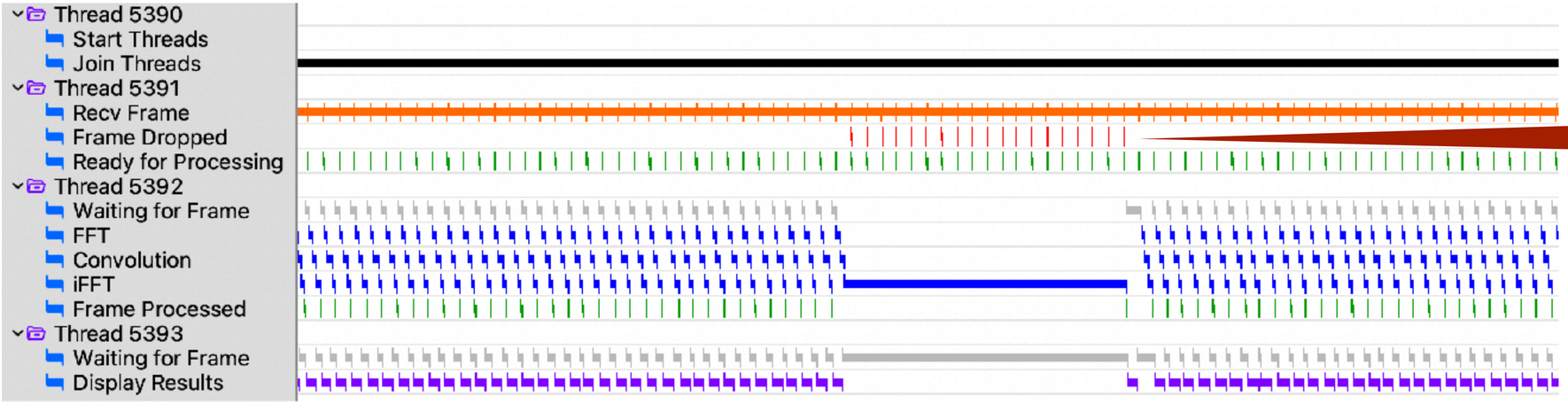
With Unikorn: Determinism

Expected steady state, **across multiple concurrent threads**



Time goes from left to right

OS may have allowed some other process to run during the FFT, resulting in dropped data

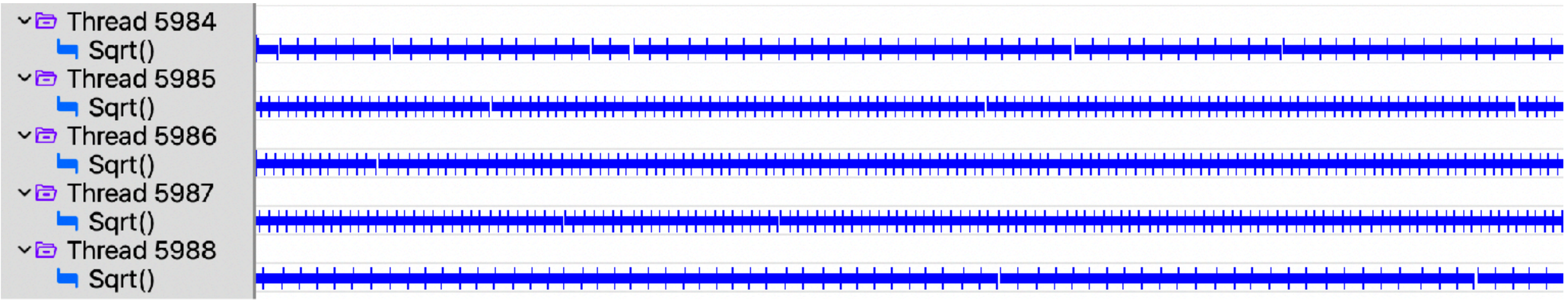


Droped Data

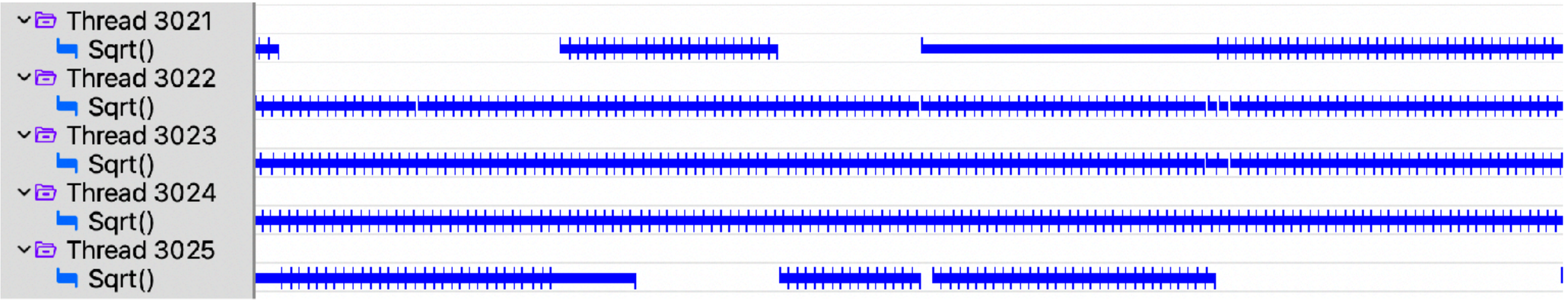
May need to use a real-time priority to avoid the mysterious stall

With Unikorn: Thread Contention/Starvation

4 cores, 5 threads, with hyper-threading

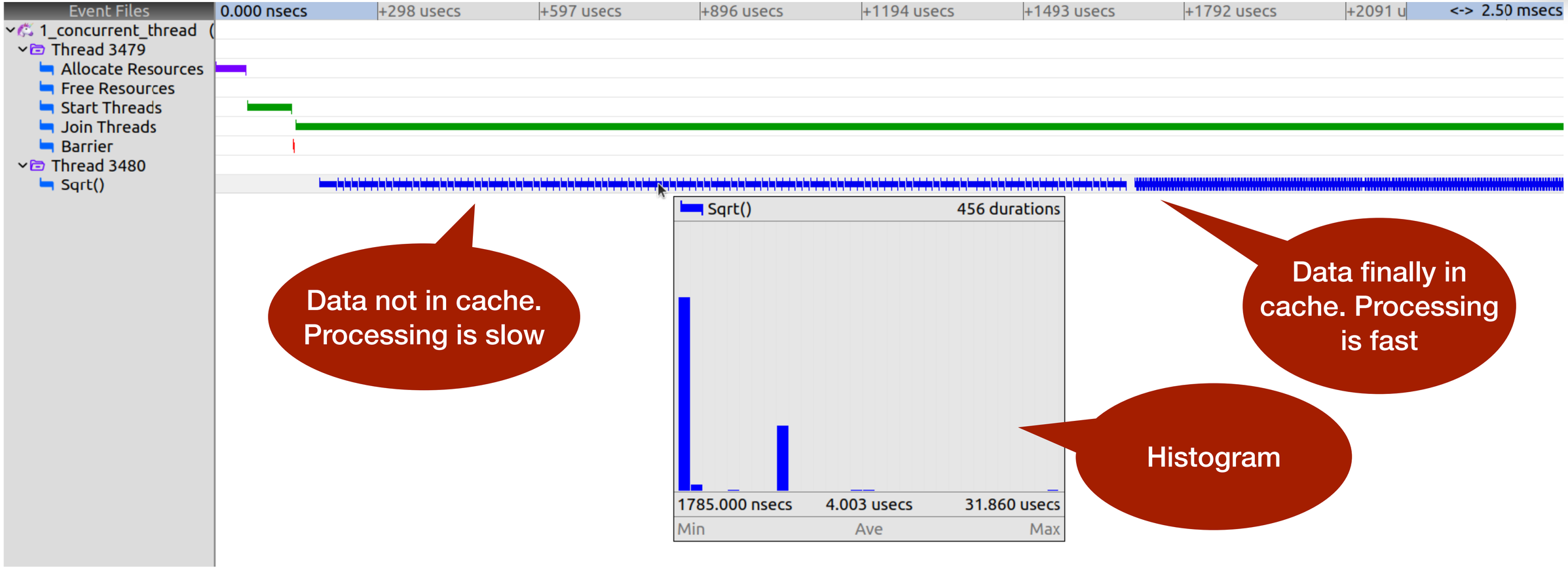


4 cores, 5 threads, without hyper-threading



Performance is no different, but hyper threading is more deterministic

With Unikorn: RAM and Cache



Initially memory accesses are cache misses, and eventually all in cache, then performance increases dramatically

How Does Unikorn Work?

Only takes a few minutes, and will result in a big productivity boost

Step 1

Instrument the source code with strategic start and end events

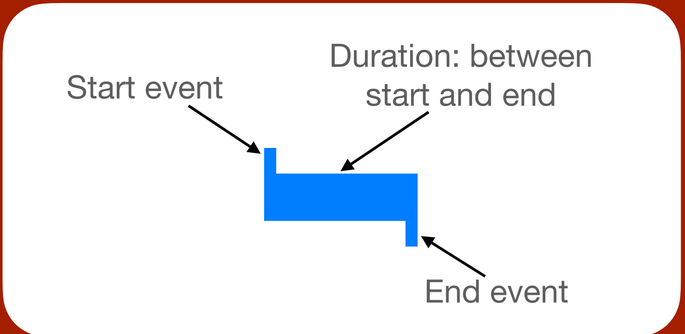
```
UNIKORN_START_PRINT(session);
printf("Hello World!\n");
UNIKORN_END_PRINT(session);
```

Step 2

Build and run the application

Step 3

View the results



Unikorn API

6 C functions, with the flexibly experts demand

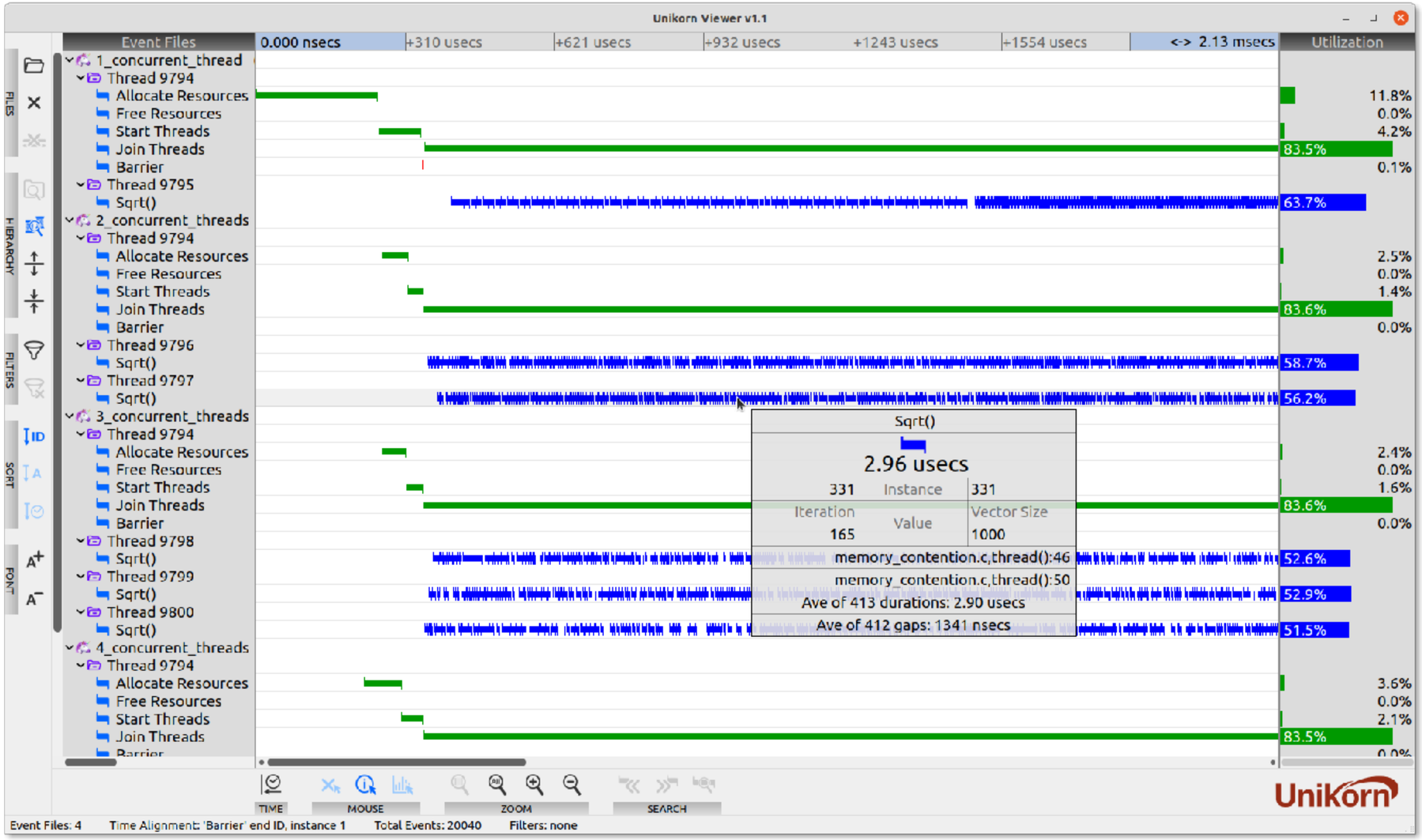
Function	Description
ukCreate()	Create a recording session
ukDestroy()	Destroy the recording session
ukRecordEvent()	Record an event
ukOpenFolder()	Put following events in specified folder
ukCloseFolder()	Pop out of the current folder
ukFlush()	Flush the stored events to the session's defined container (e.g. file, socket)

Key Notes

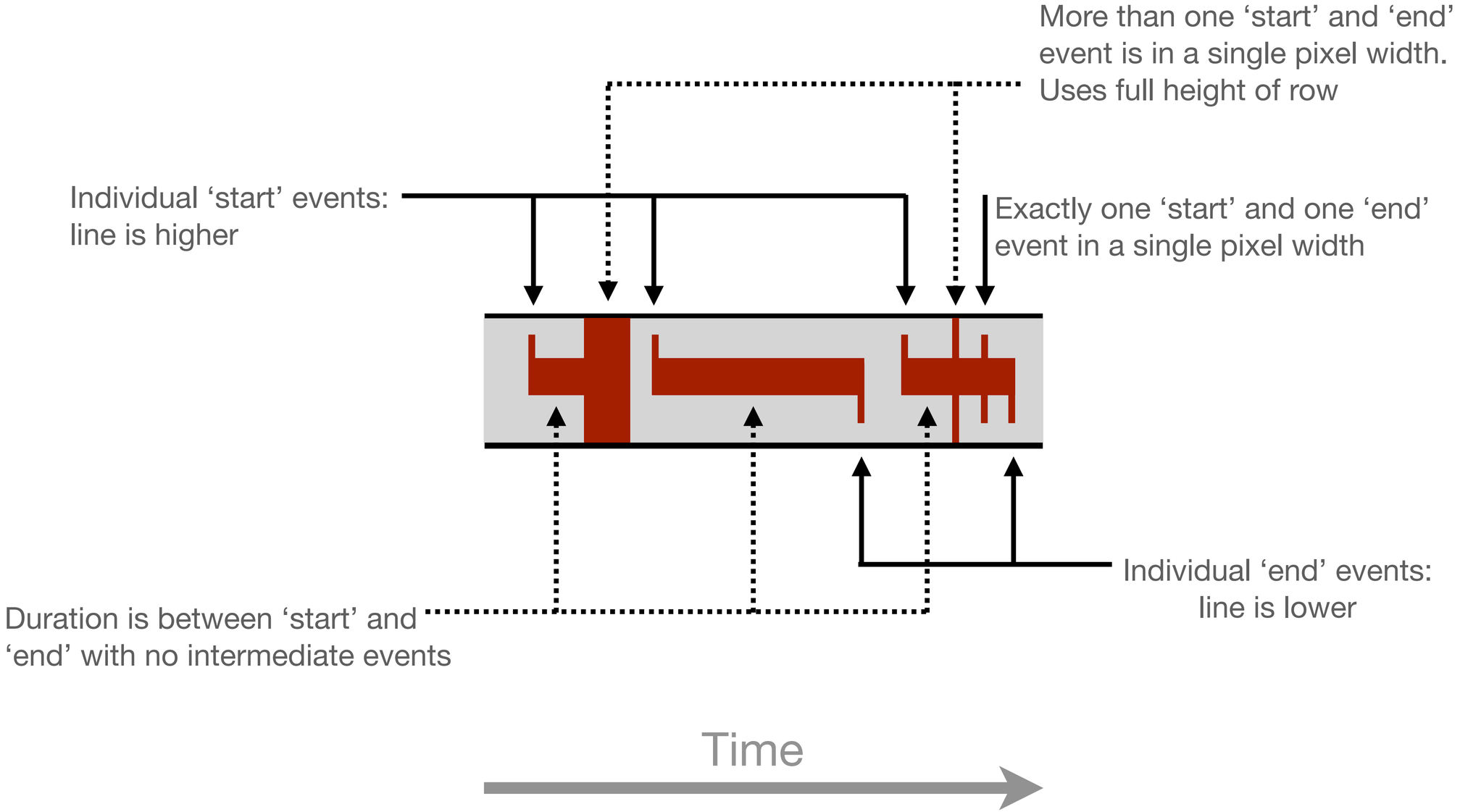
- Event names and colors are application defined: this is critical to being able to interpret results with ease
- Experts can bring their own clock (local, global, internet)
- Experts can bring their own container to flush events into (file, socket, etc.)
- Unikorn API is provided as source code (~700 SLOC), not a pre-compiled library

Unikorn Visualizer

Built for Mac, Linux, Windows



Unikorn Visualizer: Interpreting Events



Unikorn: Preparing for Instrumentation

Copy this file to your project: "**examples/hello/unikorn_instrumentation.h**"

- The goals of this file:
    1. Define the folder and event IDs (integers starting with 1)
    2. Define the folder and event registrations (IDs, names, colors, etc.)
    3. Session attributes (event buffer size, registrations, what gets recorded, etc.)
    4. Flush functionality (prepare, flush, finish. File flushing provided with Unikorn)
    5. Define the clock (to time stamp the events. Four variations provided with Unikorn)
    6. Create the session
    7. Define the open folders, record events, flush and destroy macros

Define 'ENABLE_UNIKORN_RECORDING' in the app to enable the instrumentation

Unikorn: Instrumenting Your App

Instrument any file of your application: #include “unikorn_instrumentation.h”

See “examples/hello/hello.c” as a reference

Create a new session

```
#define ENABLE_UNIKORN_SESSION_CREATION  // Only define this in the file where the sessions are created
#include "unikorn_instrumentation.h"     // Include this in any file that records events
…
#ifdef ENABLE_UNIKORN_RECORDING
  UkFileFlushInfo flush_info; // Needs to be persistent for life of session
  void *unikorn_session = UNIKORN_INIT("./hello.events", 10000, false, false, true, true, true, &flush_info);
#endif
```

Open and close folders as needed

```
UNIKORN_OPEN_FOLDER_SOLAR_SYSTEM(unikorn_session);
. . .
UNIKORN_CLOSE_FOLDER(unikorn_session);
```

Record start/end events as needed

```
UNIKORN_START_PRINT(unikorn_session, 0);
printf("Hello!\n");
UNIKORN_END_PRINT(unikorn_session, 42);
```

Flush at non-critical times

```
UNIKORN_FLUSH(unikorn_session);
```

Destroy the session when done

```
UNIKORN_FINALIZE(unikorn_session);
```

What's Recorded With Each Event?

Items	Details
ID	If it's a event (not a folder) then it will be either the start or end ID
Time Stamp	This will come from the clock function passed to ukCreate()
User Defined Value	This will be casted to a 64 bit double This is only recorded if the session was created with UkAttr.record_value == true
Instance	A counter to let you know how many times this event has been recorded This is only recorded if the session was created with UkAttr.record_instance == true
Filename Function Line number	This will let you know where in the source code the event was recorded This is only recorded if the session was created with UkAttr.record_file_location == true
Thread ID	The thread ID, as provided by the OS This is only recorded if the session was created with UkAttr.is_threaded == true

Overhead With Recording an Event

Memory

Items	Bytes
ID	2
Time Stamp	8
User Defined Value	8 (if enabled)
Instance	8 (if enabled)
Filename Function Line number	8 (if enabled) 8 (if enabled) 2 (if enabled)
Thread ID	8 (if enabled)

Time

The time to record an event will be based on the clock used and the amount of information recording that is enabled.

If multi-threaded is enabled, that will require a mutex lock for each recorded event.

To precisely know the overhead, at the beginning of your application, run some code similar to:

```
for (int i=0; i<1000; i++) {
  UNIKORN_START_OVERHEAD(session);
  UNIKORN_END_OVERHEAD(session);
}
```

IMPORTANT: Flushing is high overhead and should not be done in critical code

Other Event Analyzers

Currently there is no widely accepted open or de-facto standard

Why not?

- Seasoned developers rely on wisdom, printf, debuggers, and profilers and may not realize they are developing in the dark or at a slow pace
- Complex multi-stream software was not common in the past
- Some analyzers have a high cost
- Some analyzers are very difficult to use because they are based on OS events, not application specific events
- Some great analyzers are locked to a specific domain or product (e.g. NVIDIA's NSight, Google Chrome's Trace)

Why Choose Unikorn?

Something needs to lead the way to a de-facto standard

Considerations:

- Simple and intuitive (built on over 20 years of experience)
- It's free (Apache 2.0 license)
- Use it to create greener software (leading to reduced SWaP: Size, Weight, & Power)

Make Unikorn part of your edit, build, and test cycle

Ultimate Goal: Open Standard

Benefits if Unikorn were an Open Standard:

- API could be built into all major OSes
- Visualizer could be built into all major web browsers… no need for 3rd party tool installations. Similar to Google Chrome's Trace
- Provide consistency in interpreting visual results across companies and customers.
- Colleges and universities could teach a good habit of daily use; like with debuggers
- Globally promote greener and higher quality software design

KHRONOS® GROUP

CONNECTING SOFTWARE TO SILICON

Khronos.org (OpenGL, OpenCL, Vulkan, etc.) is an obvious candidate. It fits with their mission statement and compliments all of their open standards.