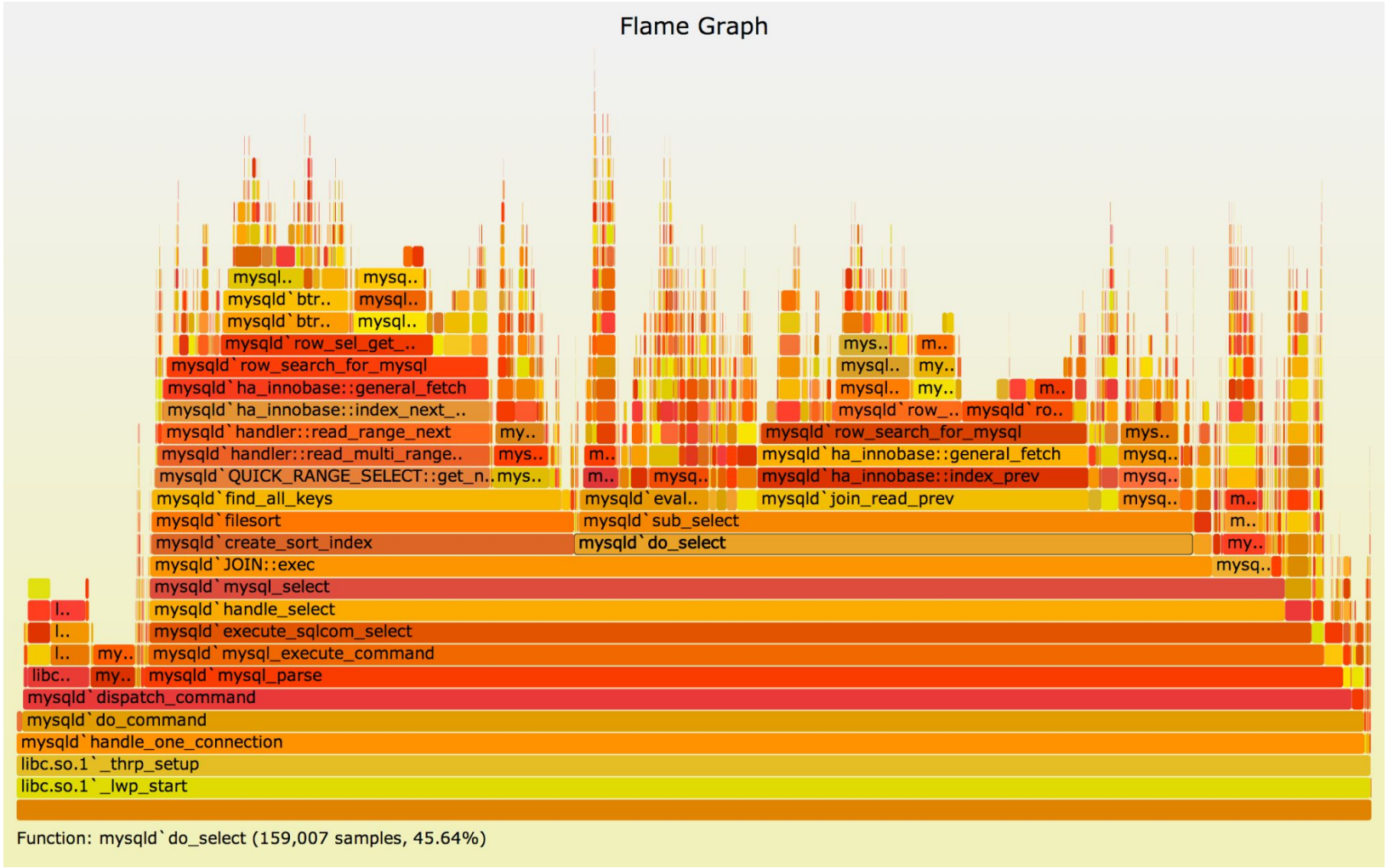
# Visualizing Execution with FlameGraphs

# What you learn in the this session

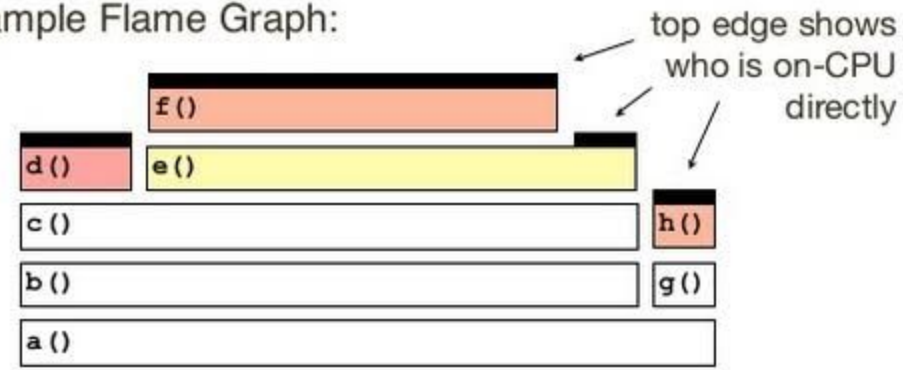
- What FlameGraphs are and how to read them
- How to generate them
- What the performance impact is on supporting them
- How to generate them from inside a container


Flame Graph
mysql..
mysq..
mysqld`btr..
mysql..
mysqld`btr..
mysql..
mysqld`row_sel_get_..
mysqld`row_search_for_mysql
mysqld`ha_innobase::general_fetch
mysqld`ha_innobase::index_next_..
mysqld`handler::read_range_next
my..
mysqld`handler::read_multi_range..
mys..
mysqld`QUICK_RANGE_SELECT::get_n..
mys..
mysqld`find_all_keys
mysqld`filesort
mysqld`create_sort_index
mysqld`JOIN::exec
mysqld`mysql_select
mysqld`handle_select
mysqld`execute_sqlcom_select
mysqld`mysql_execute_command
mysqld`mysql_parse
mysqld`dispatch_command
mysqld`do_command
mysqld`handle_one_connection
libc.so.1`_thrp_setup
libc.so.1`_lwp_start
mysqld`eval..
mysqld`sub_select
mysqld`do_select
mysqld`row_search_for_mysql
mysqld`ha_innobase::general_fetch
mysqld`ha_innobase::index_prev
mysqld`join_read_prev
mysqld`row_..
mysqld`ro..
mysql..
my..
mysq..
mys..
m..
Function: mysqld`do_select (159,007 samples, 45.64%)

- Invented by Brendan Gregg  (Sun, Oracle, Netflix, Intel) 2011
  - Presented at LISA 2013 https://www.brendangregg.com/Slides/LISA13_Flame_Graphs.pdf
- Picture of relative time spent in a single image
- Visual, navigable, obvious
  - https://raw.githubusercontent.com/brendangregg/FlameGraph/master/example-perf.svg
- Low overhead (you will measure this!)

# Flame Graphs: How to Read

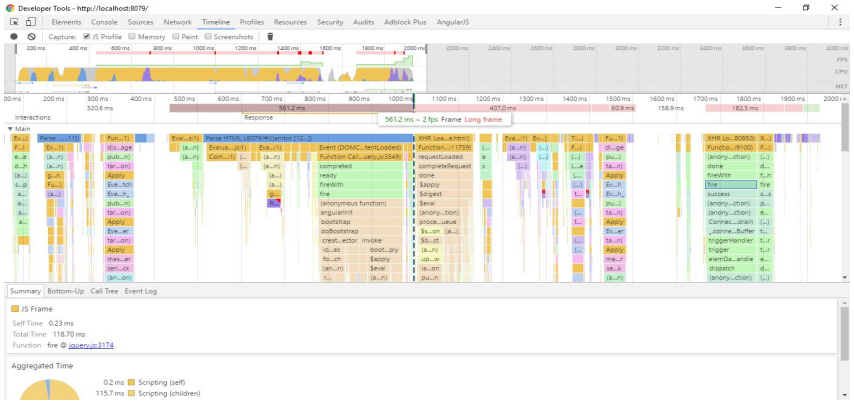
- A CPU Sample Flame Graph:

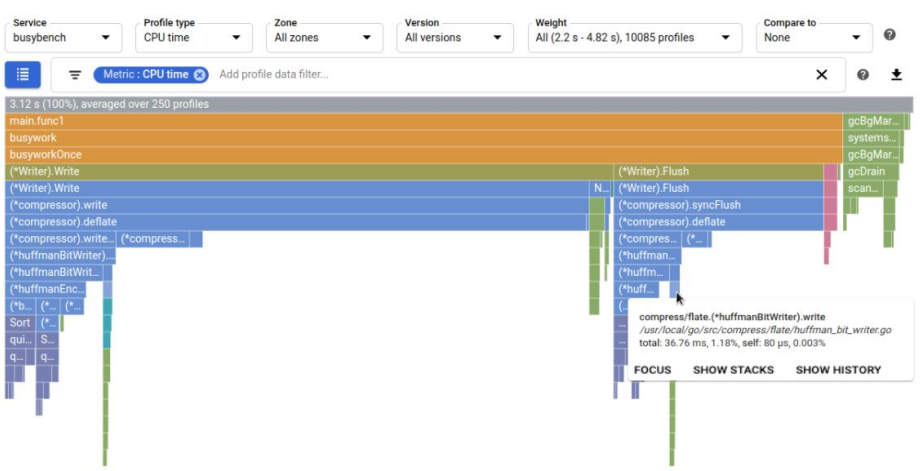



- Q: which function is on-CPU the most?
- A: `f()`

`e()` is on-CPU a little, but its runtime is mostly spent in `f()`, which is on-CPU directly

https://netflixtechblog.com/java-in-flames-e763b3d32166





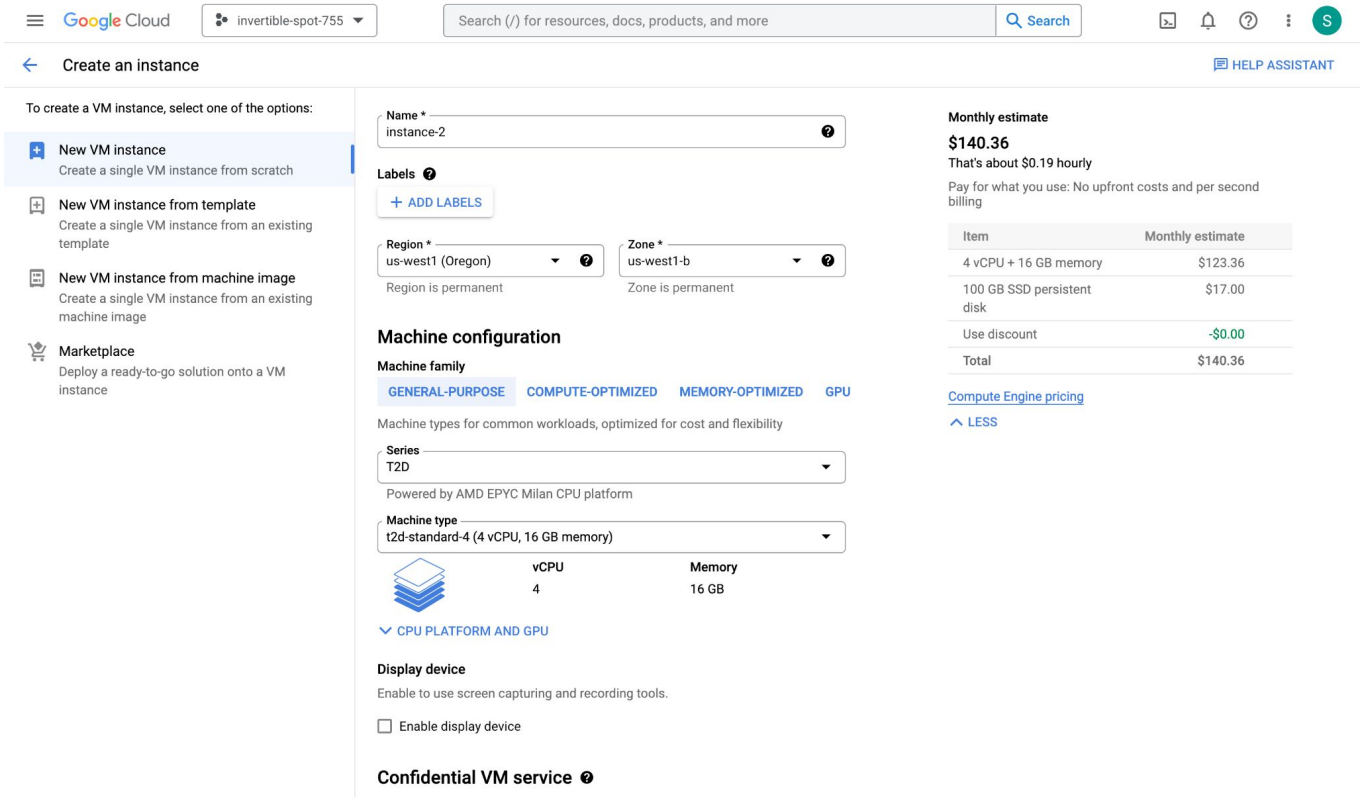

- Broad Industry Support
  - Chrome Devtools FlameCharts* (x axis is time, not aggregated counts)
    - https://developer.chrome.com/docs/devtools/performance/reference/
  - Google Cloud Profiler (IcicleGraph), rendered from the top down
    - https://cloud.google.com/profiler/docs/measure-app-performance
    - https://cloud.google.com/profiler/docs/concepts-flame
      - Built-in support for Java, Python, Node and Go
  - Grafana
    - Continuous Profiling Tool https://github.com/grafana/phlare (easily deploys on K8S)
  - Supported on everywhere
    - Windows, OSX, FreeBSD
  - Supported in every popular language

# Great Documentation

- USENIX ATC '17: Visualizing Performance with Flame Graphs
  - https://www.youtube.com/watch?v=D53T1Ejig1Q
- LISA13 - Blazing Performance with Flame Graphs
  - https://www.youtube.com/watch?v=nZfNehCzGdw
- https://github.com/brendangregg/FlameGraph
- https://www.brendangregg.com/flamegraphs.html

- Stack Sampling, sample as often or as little as your application dictates
  - `perf record -F 100 -g -- <application under measurement>`
- Can be generated by **any** profiler that captures stack back traces
  - Perf, Dtrace, Instruments, SystemTap, VTune, ktap, Xperf
  - Better quality stack information, better FlameGraphs compile binaries with
    - `-g -fno-omit-frame-pointer`
    - Fedora enabling by default in 38
      - https://fedoraproject.org/wiki/Changes/fno-omit-frame-pointer
      - Brendan Gregg wants on by default in GCC
- Install Linux perf command
  - `apt install linux-tools-$(uname -r) linux-tools-common`
  - When running in a container, tools in container must match host kernel!!

# Create a VM on the Google Cloud Console



# VM Recommendations

- Ubuntu 22.04 LTS
- t2d-standard-4 (4 physical cores, 16GB)
- 20GB+ PD-SSD for boot disk (IOPs scale with disk size)
- Allow HTTP and HTTPS (only for zero risk, ephemeral VMs)

# First Exercise

- Follow the README in this repo https://github.com/jensengrey/flamegraph-container

# Second Exercise

- Same VM or new VM
  - Take a VM Snapshot?
  - Brand new VM?
- Building and running two programs
  - Small Obfuscated Ray Tracer
  - POV-Ray

# Prepare your VM

```
# apt based (Ubuntu/Debian)
sudo sh -c 'apt update -y && apt upgrade -y'
sudo apt install build-essential -y
sudo apt install make tmux tree htop imagemagick nano vim git cmake -y
sudo apt install python3-dev python3-venv

# homebrew, https://brew.sh/
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# rust, https://rustup.rs/
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

# Create Python Environment

```
#!/bin/bash

set -eux;

python3 -m venv test.env
source test.env/bin/activate
pip install -U pip setuptools wheel
```

# Install Perf Command

```
sudo apt install linux-tools-$(uname -r) linux-tools-common -y
```

# Install FlameGraph Tools

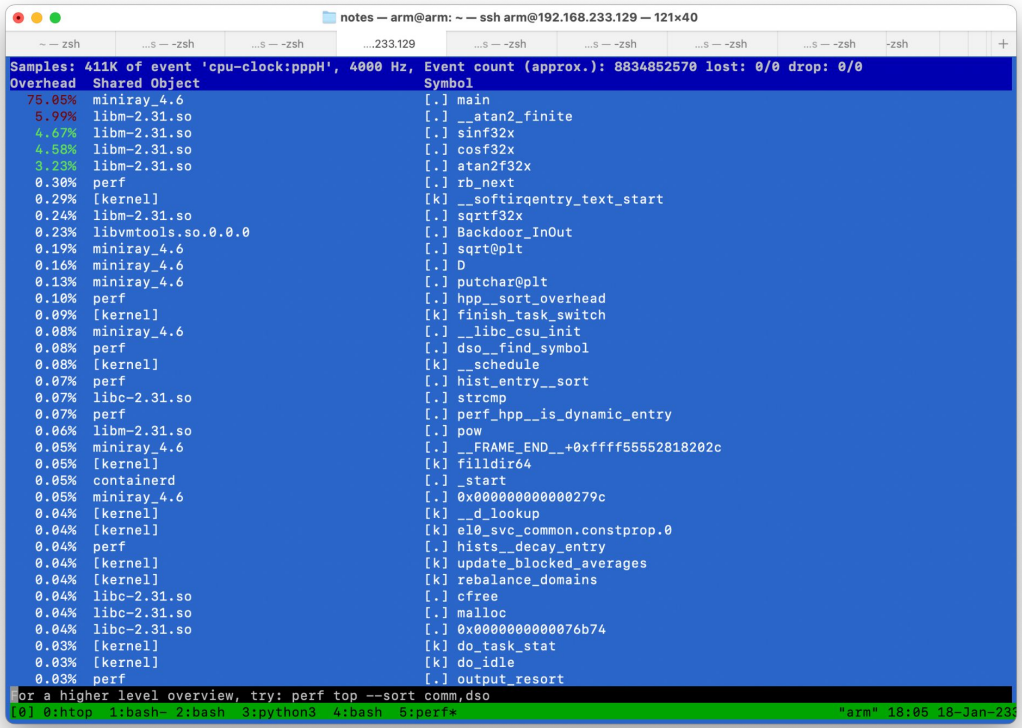
```
# the canonical tool for converting perf traces to svg
git clone https://github.com/brendangregg/FlameGraph

# more performant in some scenarios
```

- https://github.com/flamegraph-rs/flamegraph
  - https://github.com/jonhoo/inferno

# Enable sudo-less event collection

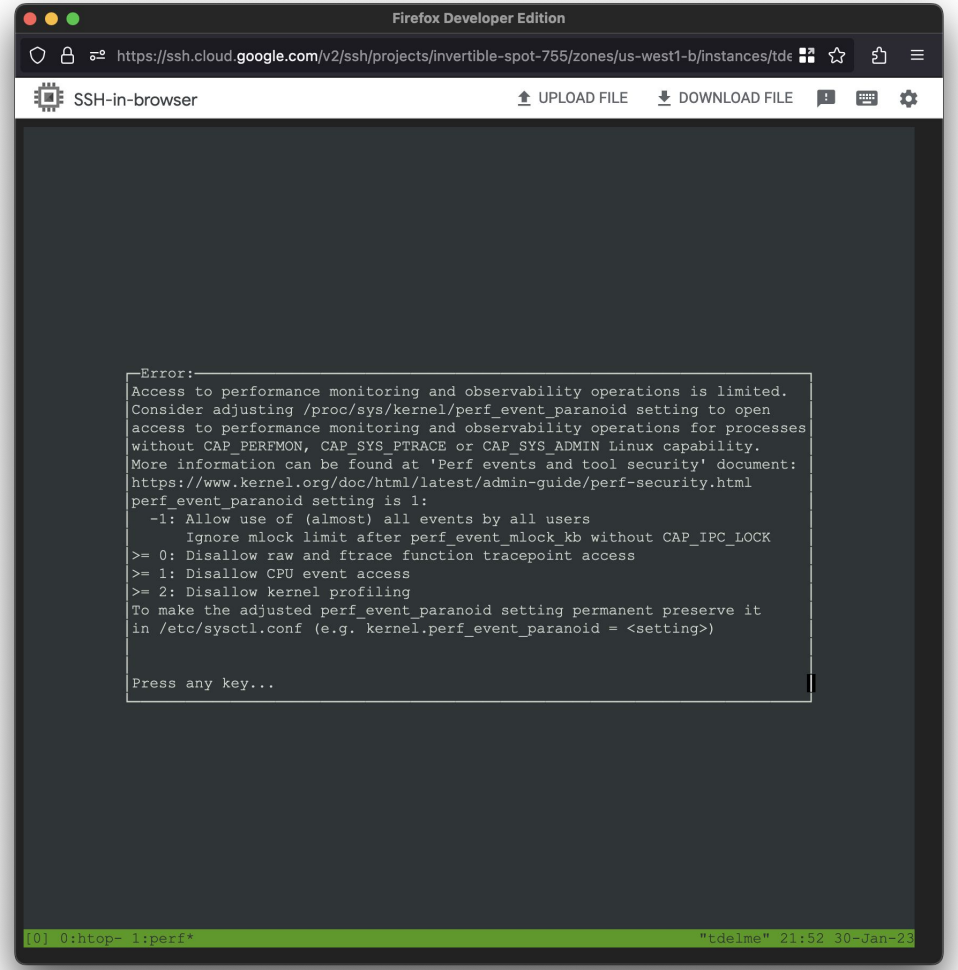
```
sudo sh -c "echo 0 > /proc/sys/kernel/kptr_restrict"
sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"
```

# perf top -e cpu-clock



# What you see when perf is restricted

```
sudo sh -c "echo 1 > /proc/sys/kernel/kptr_restrict"
sudo sh -c "echo 1 > /proc/sys/kernel/perf_event_paranoid"
```


Firefox Developer Edition
https://ssh.cloud.google.com/v2/ssh/projects/invertible-spot-755/zones/us-west1-b/instances/tde
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE
Error:
Access to performance monitoring and observability operations is limited.
Consider adjusting /proc/sys/kernel/perf_event_paranoid setting to open
access to performance monitoring and observability operations for processes
without CAP_PERFMON, CAP_SYS_PTRACE or CAP_SYS_ADMIN Linux capability.
More information can be found at 'Perf events and tool security' document:
https://www.kernel.org/doc/html/latest/admin-guide/perf-security.html
perf_event_paranoid setting is 1:
-1: Allow use of (almost) all events by all users
Ignore mlock limit after perf_event_mlock_kb without CAP_IPC_LOCK
>= 0: Disallow raw and ftrace function tracepoint access
>= 1: Disallow CPU event access
>= 2: Disallow kernel profiling
To make the adjusted perf_event_paranoid setting permanent preserve it
in /etc/sysctl.conf (e.g. kernel.perf_event_paranoid = <setting>)
Press any key...
[0] 0:htop- 1:perf*
"tdelme" 21:52 30-Jan-23

# Enable sudo-less event collection

```
sudo sh -c "echo 0 > /proc/sys/kernel/kptr_restrict"
sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"
```

Call graph recording

Samples per second

```
perf record -F 100 -a -g -- <application under measurement>
```

All cores

# First Test Workload

```
git clone https://github.com/mzucker/miniray
cd miniray
mkdir build
```

Edit ../CmakeLists.txt

```
cmake_minimum_required(VERSION 2.8.12)

set(GCC_COVERAGE_COMPILE_FLAGS "-g3 -fno-omit-frame-pointer")

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${GCC_COVERAGE_COMPILE_FLAGS}")
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${GCC_COVERAGE_COMPILE_FLAGS}")

project(miniray)

set(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR})

add_subdirectory(src)
```

## Run cmake

```
 cmake ..
-- The C compiler identification is GNU 11.3.0
-- The CXX compiler identification is GNU 11.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/sean_jensengrey/miniray/build
```

# Build the code

```
$ make
[  3%] Built target encode
[  6%] Built target miniray_4.6
[ 10%] Built target miniray_4.5
[ 13%] Built target miniray_1.3
[ 16%] Built target miniray_4.6_commented
[ 20%] Built target miniray_4.0
[ 23%] Built target miniray_2.4
...
# src/miniray_0.2.cpp:73:3: error: narrowing conversion of ‘2169135176’ from
‘unsigned int’ to ‘i’ {aka ‘int’} [-Wnarrowing]
```

# Render First Image

```
$ perf stat ./miniray_4.6 > test2.ppm

 Performance counter stats for './miniray_4.6':

         17,380.65 msec task-clock                #    0.998 CPUs utilized
         2,498     context-switches          #    0.144 K/sec
             3     cpu-migrations            #    0.000 K/sec
             52    page-faults               #    0.003 K/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     17.415531414 seconds time elapsed

     17.378465000 seconds user
     0.000000000 seconds sys
```

# Render First Image

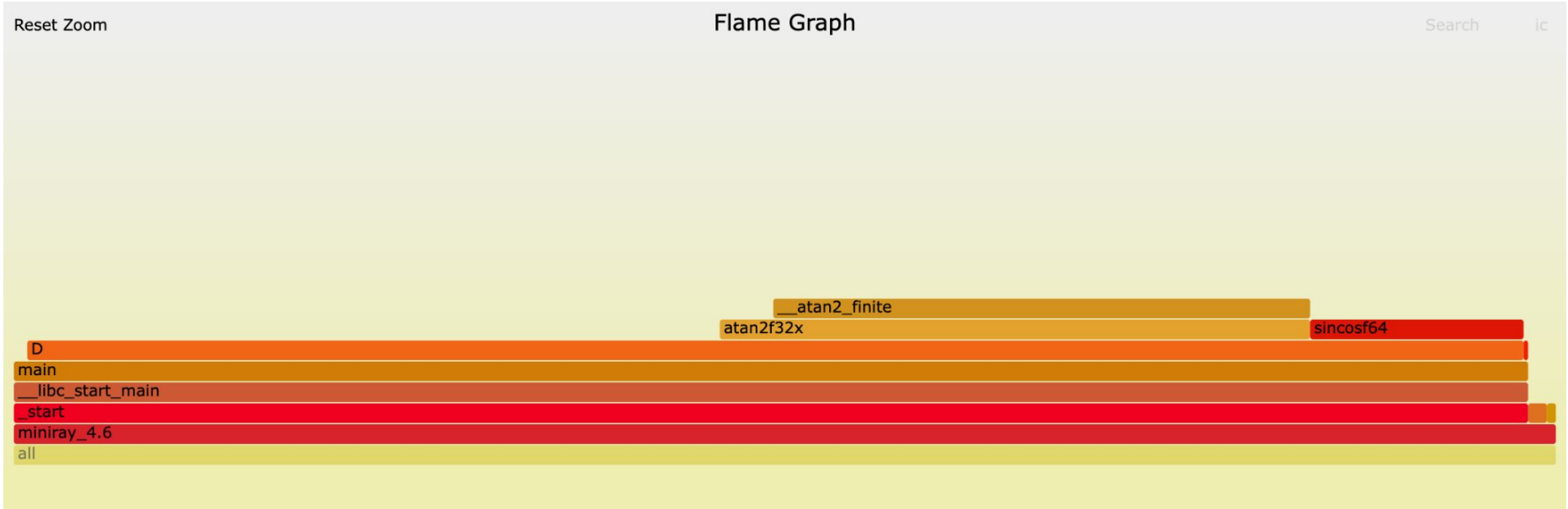
```
$ perf stat ./miniray_4.6 > miniray_46.ppm
$ convert miniray_46.ppm miniray_46.jpg
```

# Peruse File System

```
python3 -m http.server 9999
# get your external ip
https://www.google.com/search?hl=en&q=my%20ip%20address
# add a firewall rule for your IP, allow all
gcloud compute --project=<your project> \
    firewall-rules create devaccess \
    --direction=INGRESS \
    --priority=400 \
    --network=default \
    --action=ALLOW \
    --rules=tcp \
    --source-ranges=<your ip> \
    --enable-logging
```

```
$ perf record -F 10000 -g -- ./miniray_4.6 > miniray_4.6.ppm
```

```
$ cd ~/FlameGraph;
$ perf script | ./stackcollapse-perf.pl |./flamegraph.pl > perf.svg
```



# Exercise 1: Generate FlameGraph

Follow the instructions on the previous slide and generate the flamegraph.svg

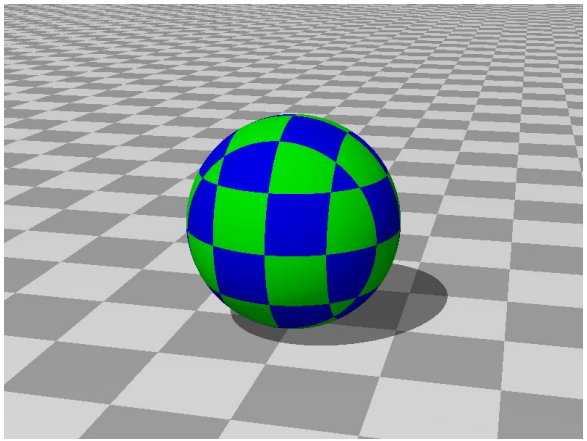
# Exercise 2: What is the impact on wall clock time for including debug information?

# Install Docker on Ubuntu

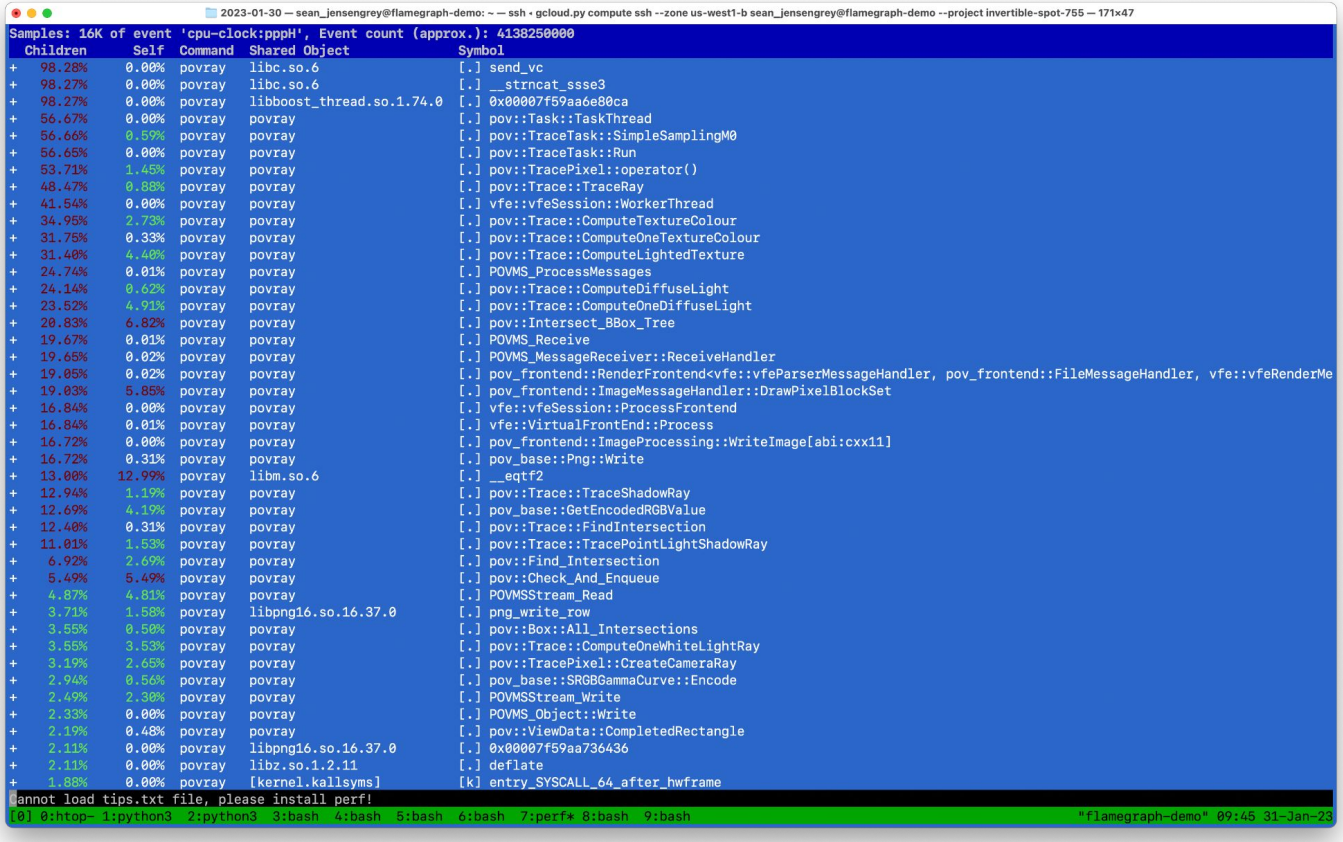
- Bring up an Ubuntu VM
- Install Docker
- Install devtools

# Build Container and Render Test Image

```
git clone https://github.com/jensengrey/workshop-containers-af99
cd workshop-containers-af99/povray
sudo docker build -t povtest:1 - < povray.dockerfile
sudo docker run -t --rm -v $PWD:/povfiles --user $(id -u):$(id -g) povtest:1 povray
/povfiles/debug.pov
```



# perf report --no-inline



# Hacking Dockerfiles

- Don't suffer in docker build loop
- Spin up a VM with the same base image as your container, debug and then port into Dockerfile

# Single Best Docker+Perf Tutorial

- https://gendignoux.com/blog/2019/11/09/profiling-rust-docker-perf.html

# Docker Locks Down Specific Syscalls

- https://docs.docker.com/engine/security/seccomp/

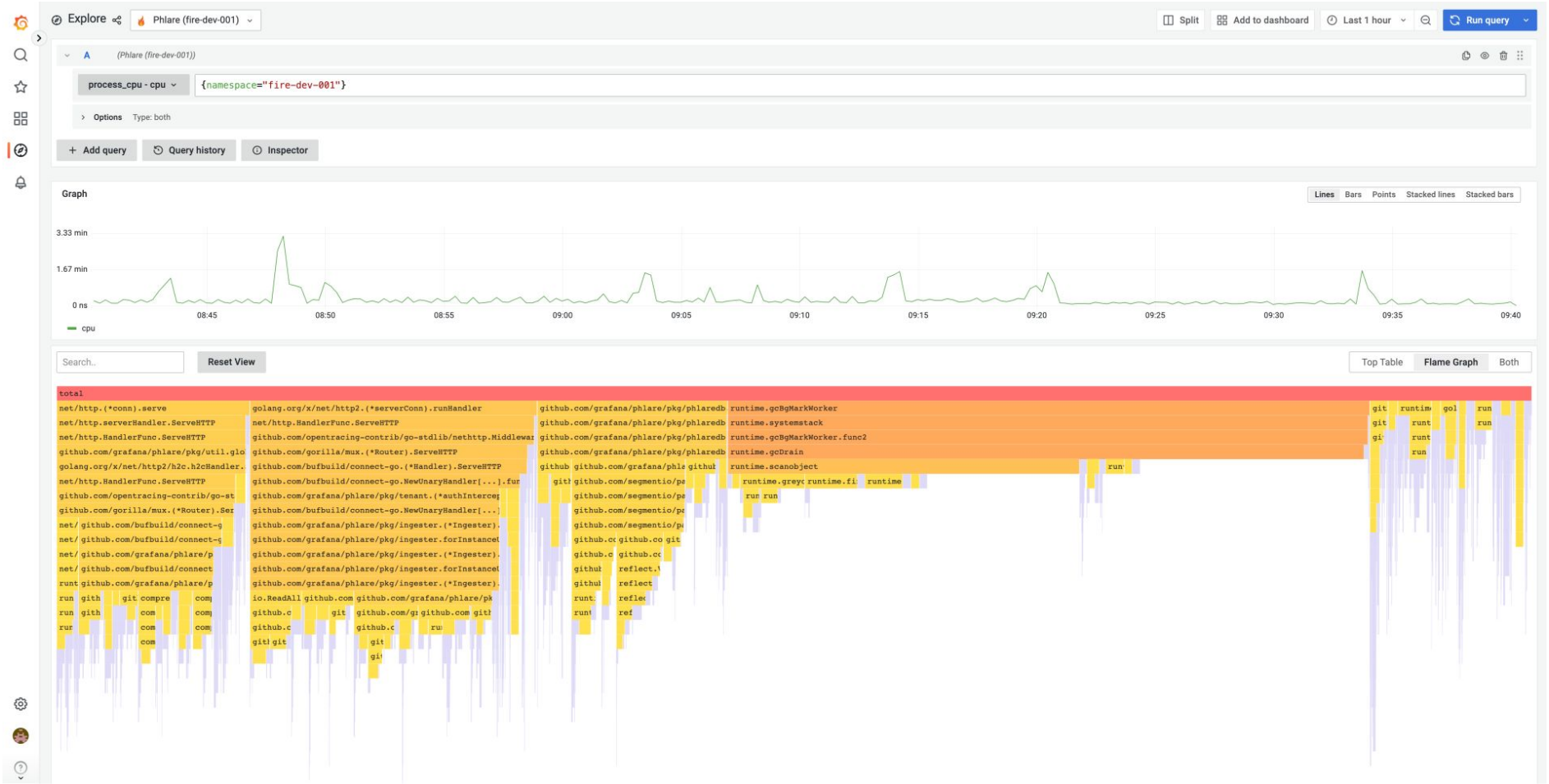
# Modify Build Command to Support Perf

```
cd povray/unix && \
     ./prebuild.sh && \
     cd .. && \
     CXXFLAGS="-fno-omit-frame-pointer" ./configure --enable-debug --enable-profile
COMPILED_BY="povray-perf-builder" && \
     make -j4 && \
     make install
```

# Generate Perf Trace

```
sudo docker run -t -v $PWD:/povfiles --security-opt seccomp=unconfined povtest-perf:1 perf record -F 100 -a -g -- povray /povfiles/debug.pov;
```

# Continuous Profiling in Grafana Phlare



# Grafana Phlare

- https://grafana.com/oss/phlare/
- https://github.com/grafana/phlare
- https://grafana.com/docs/phlare/latest/
- https://grafana.com/docs/phlare/latest/operators-guide/deploy-kubernetes/
- https://grafana.com/tutorials/
- https://www.youtube.com/watch?v=ORINvZAurIY