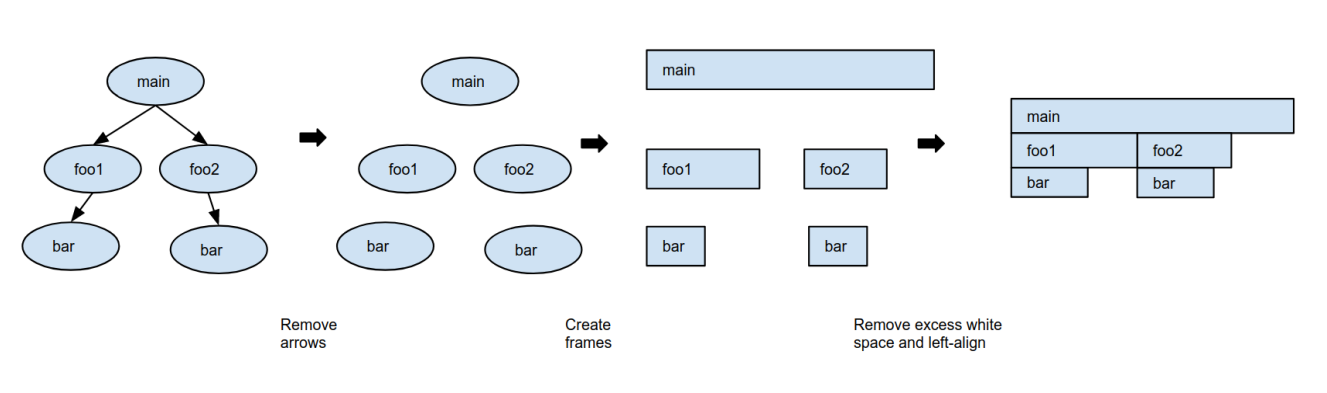
Stackdriver Profiler displays profiling data by using Flame Graphs (http://www.brendangregg.com/flamegraphs.html). Unlike trees and graphs, flame graphs make efficient use of screen space by representing a large amount of information in a compact and readable format.

To introduce flame graphs, this page illustrates how to convert a tree into a flame graph and summarizes key features of flame graphs.

To create a flame graph from a tree, complete the steps illustrated in the following diagram:



1. Remove from the tree the arrows that indicate function calls.
2. Replace each tree node with a *frame*.

   Frames are rectangular in shape and all frames have the same height. For the example on this page, the total CPU time used by the function named in the frame determines the frame width.

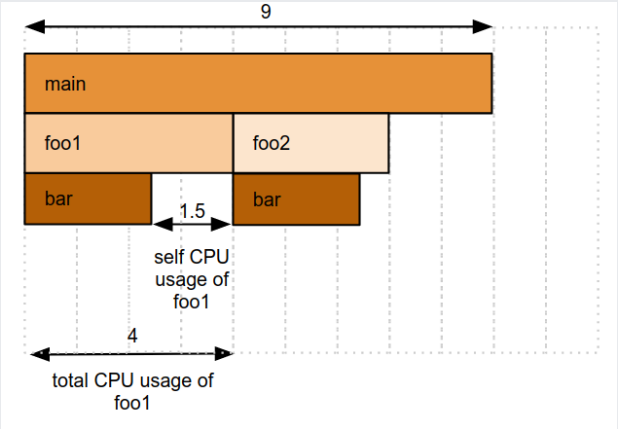
Key Point: In profiling, a functions' *total CPU time* is the CPU time used by the function including the CPU time used by all functions it calls. A function's *self CPU time* is the CPU time used by a function excluding the CPU time used by the functions it calls.

The pseudo code for each of the functions is described in the following table. The CPU intensive work performed during a function's execution defines the self CPU time:

Function pseudo code	self CPU time (seconds)	total CPU time (seconds)
	2	4 + 3 + 2 = 9
	1.5	2.5 + 1.5 = 4
	0.5	2.5 + 0.5 = 3
	2.5	2.5

3. The next step is to remove the *vertical* space between the frames and left align frames while preserving call sequences. Optionally, you can define a color scheme and color the frames according to the definition. For example, you can color frames by their package, by total CPU time, by self CPU time, or by a different measure.

   After removing excess whitespace and coloring frames by the self CPU time, the flame graph now appears as follows:



Notice that the call stacks for `foo1` and `foo2` have been preserved, even though the call stack starting with `foo2` is now next to the frame for `foo1`.

This simple example illustrates the following:

- Flame graphs are a compact representation of a tree and you can recreate a call stack by tracing frames from the top downwards.
- Frames name a function and the frame width is the relative measure of that function's total CPU time. In this example, because the total CPU time of `foo2` is one third of the total CPU time of `main`, the frame for `foo2` is one third the width of the frame for `main`.
- The width of the empty space below a frame is the relative measure of the self CPU time for the function named in the frame. For example, below the frame `foo1`, 1.5 units are empty and 2.5 units are occupied by `bar`. Therefore the self CPU time of `foo1` is 37.5% of its total CPU time, or 1.5 s.
- As you follow a call stack, the widths of the frames decrease because the total CPU time of a callee can never be more than the total CPU time of the caller. This behavior is what causes the flame shape.

  In the example, `foo1` calls `bar` and the total CPU time of `foo1` is defined to be the total CPU time of `bar` *plus* the self CPU time of `foo1`. Therefore, the total CPU time of `bar` cannot be more than the total CPU time of `foo1`.

- For information on the supported operating environments, languages, and performance impact of Stackdriver Profiler, go to About Stackdriver Profiler (/profiler/docs/about-profiler).
- For an introduction to profiling, go to Profiling concepts (/profiler/docs/concepts-profiling).
- For details on the Stackdriver Profiler interface, go to Using the Stackdriver Profiler interface (/profiler/docs/using-profiler).
- To get started, try the Stackdriver Profiler Quickstart (/profiler/docs/quickstart).