

# Speed Up JavaScript Execution



By Kayce Basques

Technical Writer for Chrome DevTools



By Meggin Kearney

Meggin is a Tech Writer

Identify expensive functions using the Chrome DevTools CPU Profiler.

Elements Console Sources Network Timeline Profiles Resources Security Audits			
Heavy (Bottom Up) ▾ 🔍 ✕ ↺			
Profiles	Self ▾	Total	Function
	4447.3 ms	4447.3 ms	(idle)
CPU PROFILES	2162.6 ms 6.61 %	2165.4 ms 6.62 %	▶ <b>montReduce</b> <a href="#">crypto.js:583</a>
Profile 1 Save	1951.8 ms 5.97 %	1951.8 ms 5.97 %	(garbage collector)
	1643.9 ms 5.02 %	1652.8 ms 5.05 %	▶ lin_solve <a href="#">navier-stokes.js:152</a>
	1476.7 ms 4.51 %	1964.1 ms 6.00 %	▶ Scheduler.schedule <a href="#">richards.js:188</a>
	1271.8 ms 3.89 %	1271.8 ms 3.89 %	(program)
	1170.8 ms 3.58 %	1172.0 ms 3.58 %	▶ bnpSquareTo <a href="#">crypto.js:431</a>
	987.9 ms 3.02 %	1081.7 ms 3.31 %	▶ GeneratePayloadTree <a href="#">splay.js:50</a>
	884.5 ms 2.70 %	2269.5 ms 6.94 %	▶ a8 <a href="#">(program):1</a>
	763.5 ms 2.33 %	837.0 ms 2.56 %	▶ one_way_unify1_nboyer <a href="#">earley-boyer.js:3635</a>
	720.7 ms 2.20 %	720.7 ms 2.20 %	▶ a6 <a href="#">(program):1</a>
	682.6 ms 2.09 %	1577.2 ms 4.82 %	▶ rewrite_nboyer <a href="#">earley-boyer.js:3604</a>
	624.5 ms 1.91 %	624.5 ms 1.91 %	▶ SplayTree.splay_ <a href="#">splay.js:322</a>
	619.2 ms 1.89 %	846.0 ms 2.59 %	▶ Exec <a href="#">regex.js:1</a>
	558.0 ms 1.71 %	795.0 ms 2.43 %	▶ (anonymous function) <a href="#">code-load.js:1518</a>
	540.0 ms 1.65 %	540.4 ms 1.65 %	▶ Plan.execute <a href="#">deltablue.js:776</a>
	517.8 ms 1.58 %	799.7 ms 2.44 %	▶ (anonymous function) <a href="#">code-load.js:1541</a>
	458.2 ms 1.40 %	1348.3 ms 4.12 %	▶ loop2 <a href="#">earley-boyer.js:4272</a>
	402.6 ms 1.23 %	402.8 ms 1.23 %	▶ HandlerTask.run <a href="#">richards.js:465</a>
	320.8 ms 0.98 %	582.2 ms 1.78 %	▶ Constraint.satisfy <a href="#">deltablue.js:175</a>
	312.6 ms 0.96 %	1333.4 ms 4.08 %	▶ loop3 <a href="#">earley-boyer.js:4286</a>
	301.8 ms 0.92 %	1348.7 ms 4.12 %	▶ sc_loop1_98 <a href="#">earley-boyer.js:4258</a>
	274.2 ms 0.84 %	1349.6 ms 4.12 %	▶ deriv_trees <a href="#">earley-boyer.js:4254</a>

## TL;DR

- Record exactly which functions were called and how long each took with the CPU Profiler.
- Visualize your profiles as a flame chart.

## Record a CPU profile

If you're noticing jank in your JavaScript, collect a JavaScript CPU profile. CPU profiles show where execution time is spent in your page's functions.

1. Go to the **Profiles** panel of DevTools.
2. Select the **Collect JavaScript CPU Profile** radio button.
3. Press **Start**.
4. Depending on what you are trying to analyze, you can either reload the page, interact with the page, or just let the page run.
5. Press the **Stop** button when you are finished.

You can also use the [Command Line API](#) to record and group profiles from the command line.


## View CPU profile

When you finish recording, DevTools automatically populates the Profile panel with the data from your recording.

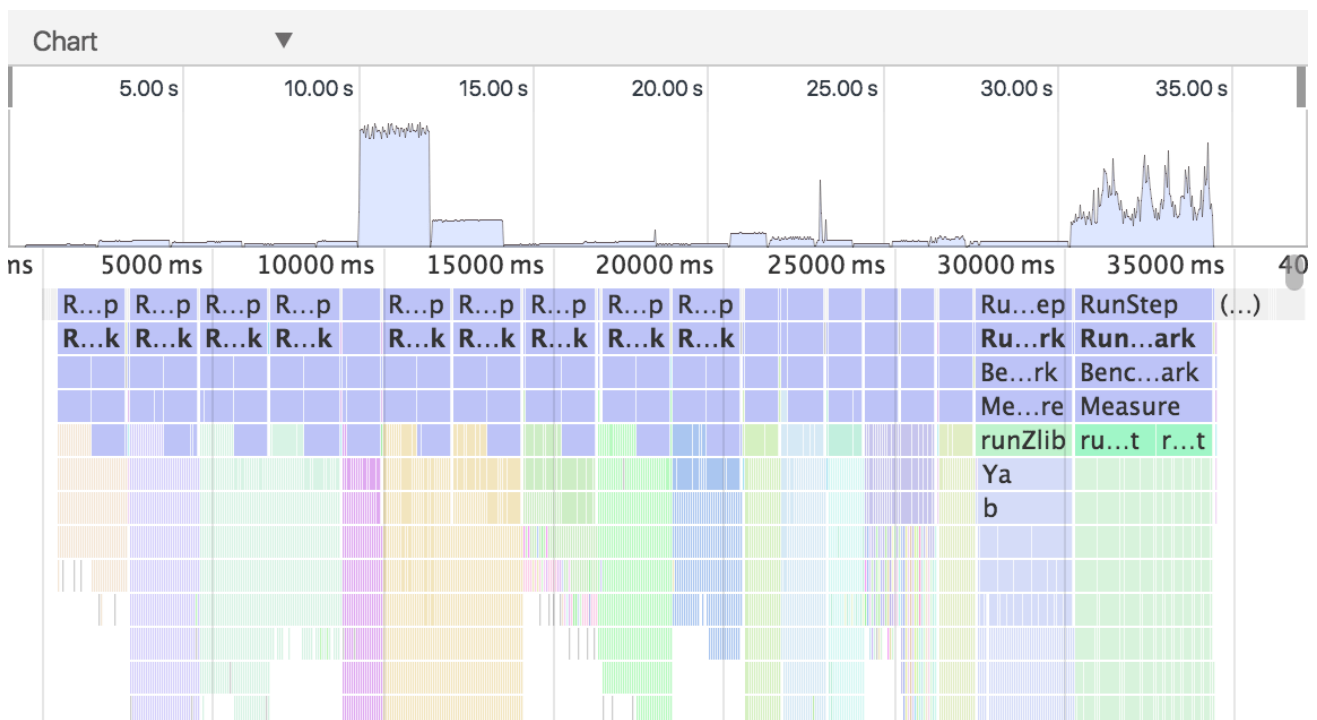
The default view is **Heavy (Bottom Up)**. This view enables you to see which functions had the most impact on performance and examine the calling paths to those functions.

## Change sort order

To change the sorting order, click on the dropdown menu next to the **focus selected function**

icon (  ) and then choose one of the following options:




**Chart.** Displays a chronological flame chart of the recording.



**Heavy (Bottom Up).** Lists functions by impact on performance and enables you to examine the calling paths to the functions. This is the default view.


Heavy (Bottom Up) <span>▼</span> <span>👁</span> <span>✕</span> <span>↺</span>					
Self	Total		Function		
4447.3 ms	4447.3 ms		(idle)		
2162.6 ms 6.61 %	2165.4 ms	6.62 %	▶ montReduce	<a href="#">crypto.js:583</a>	
1951.8 ms 5.97 %	1951.8 ms	5.97 %	(garbage collector)		
1643.9 ms 5.02 %	1652.8 ms	5.05 %	▶ lin_solve	<a href="#">navier-stokes.js:152</a>	
1476.7 ms 4.51 %	1964.1 ms	6.00 %	▶ Scheduler.schedule	<a href="#">richards.js:188</a>	
1271.8 ms 3.89 %	1271.8 ms	3.89 %	(program)		
1170.8 ms 3.58 %	1172.0 ms	3.58 %	▶ bnpSquareTo	<a href="#">crypto.js:431</a>	
987.9 ms 3.02 %	1081.7 ms	3.31 %	▶ GeneratePayloadTree	<a href="#">splay.js:50</a>	
884.5 ms 2.70 %	2269.5 ms	6.94 %	▶ a8	<a href="#">(program):1</a>	
763.5 ms 2.33 %	837.0 ms	2.56 %	▶ one_way_unify1_nboyer	<a href="#">earley-boyer.js:3635</a>	
720.7 ms 2.20 %	720.7 ms	2.20 %	▶ a6	<a href="#">(program):1</a>	
682.6 ms 2.09 %	1577.2 ms	4.82 %	▶ rewrite_nboyer	<a href="#">earley-boyer.js:3604</a>	
624.5 ms 1.91 %	624.5 ms	1.91 %	▶ SplayTree.splay_	<a href="#">splay.js:322</a>	
619.2 ms 1.89 %	846.0 ms	2.59 %	▶ Exec	<a href="#">regexp.js:1</a>	
558.0 ms 1.71 %	795.0 ms	2.43 %	▶ (anonymous function)	<a href="#">code-load.js:1518</a>	
540.0 ms 1.65 %	540.4 ms	1.65 %	▶ Plan.execute	<a href="#">deltablue.js:776</a>	
517.8 ms 1.58 %	799.7 ms	2.44 %	▶ (anonymous function)	<a href="#">code-load.js:1541</a>	


**Tree (Top Down).** Shows an overall picture of the calling structure, starting at the top of the call stack.

Tree (Top Down) ▼   					
Self ▼		Total		Function	
4447.3 ms		4447.3 ms		(idle)	
1951.8 ms	5.97 %	1951.8 ms	5.97 %	(garbage collector)	
1271.8 ms	3.89 %	1271.8 ms	3.89 %	(program)	
0.5 ms	0.00 %	29493.0 ms	90.14 %	▶ RunStep	<a href="#">base.js:147</a>
0.3 ms	0.00 %	2.2 ms	0.01 %	▶ eventHandle	<a href="#">jquery.js:2857</a>
0 ms	0 %	1.2 ms	0.00 %	▶ (anonymous function)	<a href="#">(program):1</a>

## Exclude functions

To exclude a function from your CPU profile, click on it to select it and then press the

**exclude selected function** icon (  ). The caller of the excluded function is charged with the excluded function's total time.

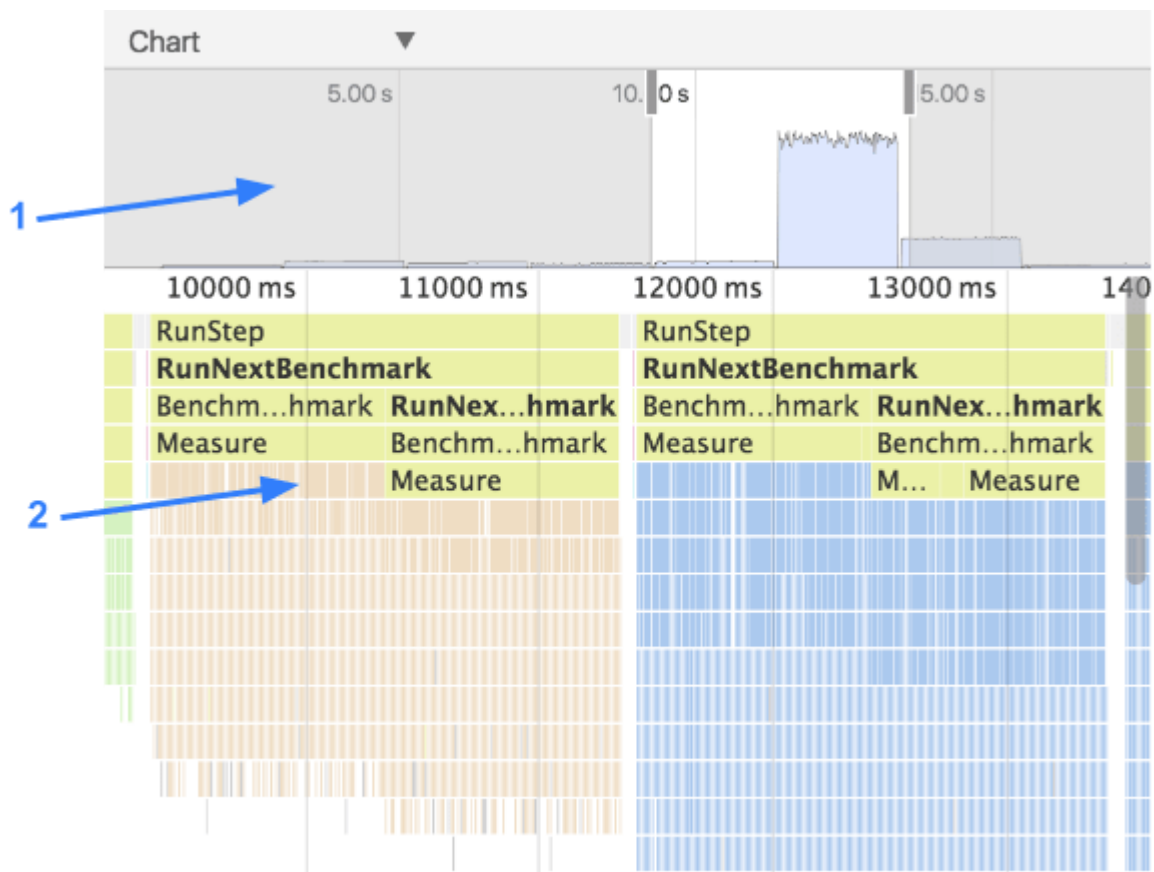
Click the **restore all functions** icon (  ) to restore all excluded functions back into the recording.

## View CPU profile as Flame Chart

The Flame Chart view provides a visual representation of the CPU profile over time.

After recording a CPU profile, view the recording as a flame chart by changing the sort order to **Chart**.

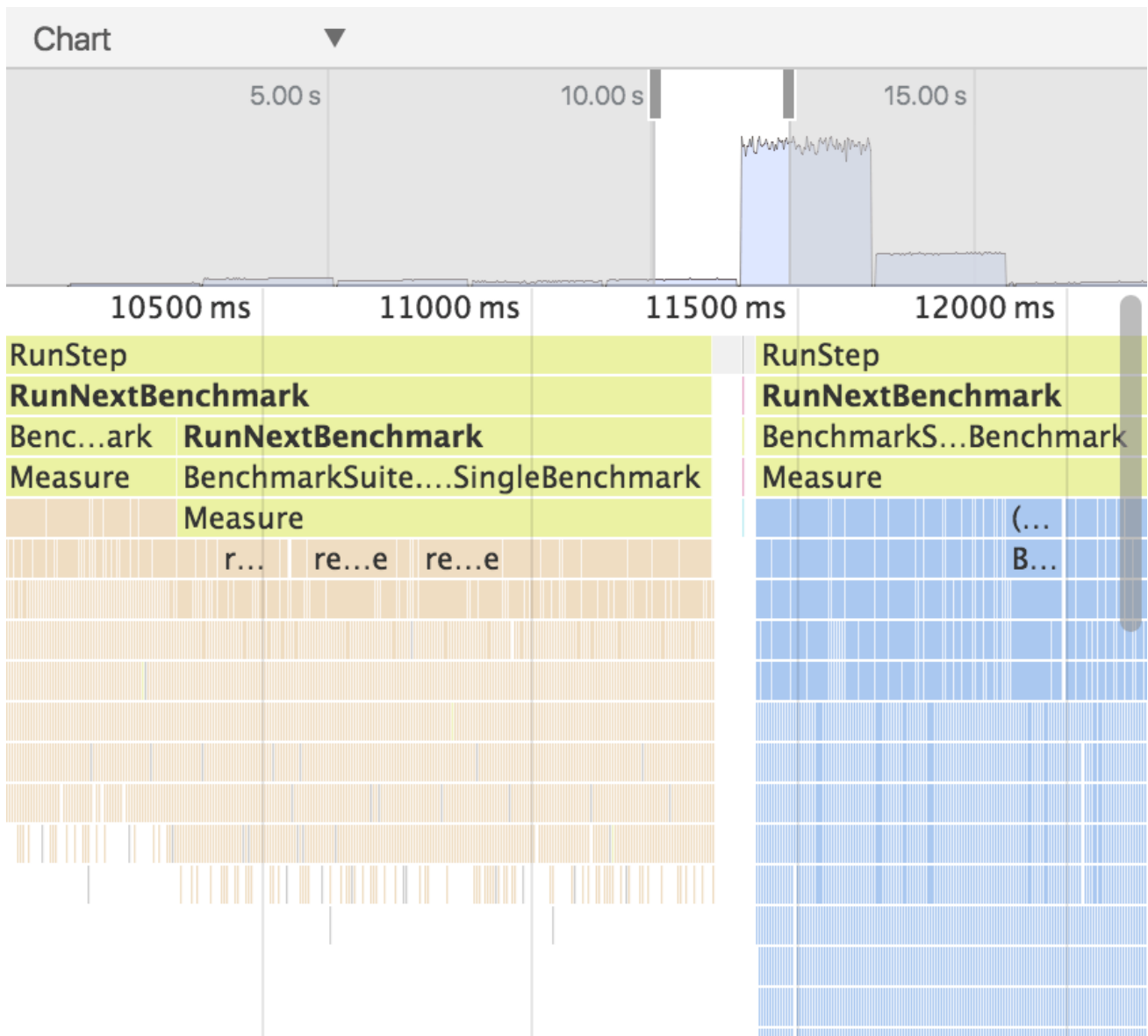




A tall call stack is not necessarily significant, it just means that a lot of functions were called. But a wide bar means that a call took a long time to complete. These are candidates for optimization.

## Zoom in on specific parts of recording

Click, hold, and drag your mouse left and right across the overview to zoom in on particular parts of the call stack. After you zoom, the call stack automatically displays the portion of the recording that you've selected.



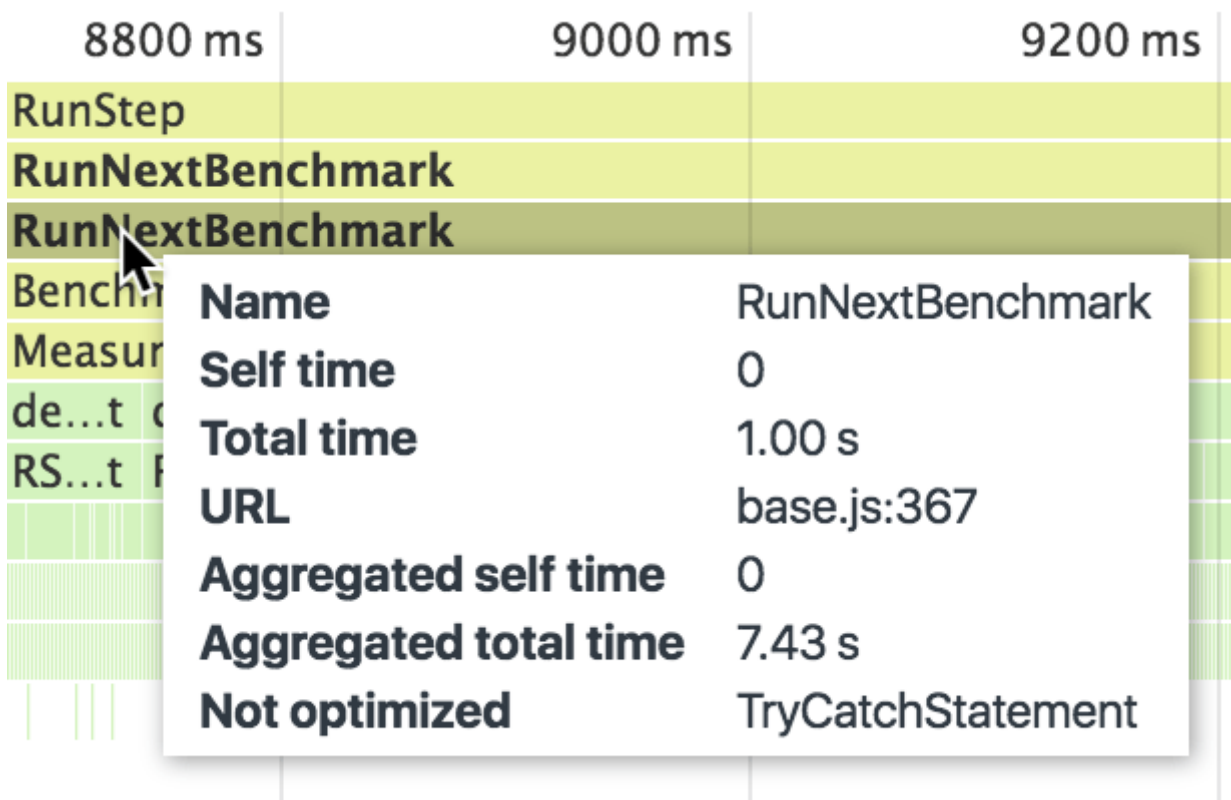
## View function details

Click on a function to view its definition in the **Sources** panel.

Hover over a function to display its name and timing data. The following information is provided:

- **Name.** The name of the function.
- **Self time.** How long it took to complete the current invocation of the function, including only the statements in the function itself, not including any functions that it called.
- **Total time.** The time it took to complete the current invocation of this function and any functions that it called.

- **URL.** The location of the function definition in the form of `file.js:100` where `file.js` is the name of the file where the function is defined and `100` is the line number of the definition.
- **Aggregated self time.** Aggregate time for all invocations of the function across the recording, not including functions called by this function.
- **Aggregated total time.** Aggregate total time for all invocations of the function, including functions called by this function.
- **Not optimized.** If the profiler has detected a potential optimization for the function it lists it here.



Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.