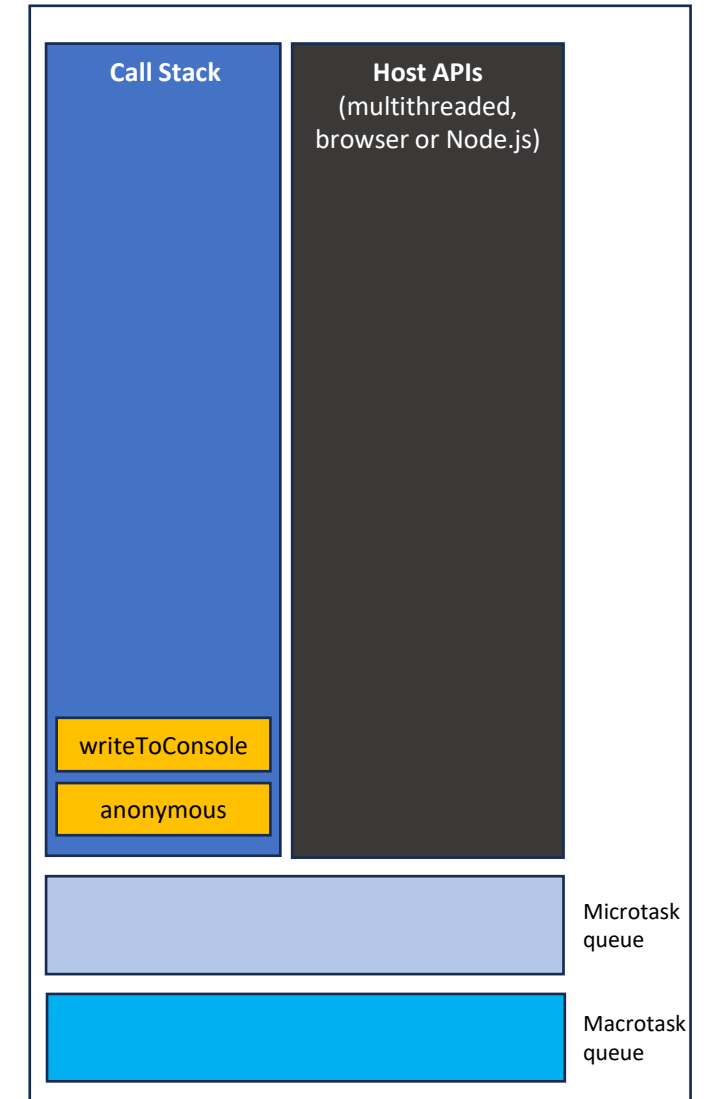


Event 1: Macrotask

```
1  #!/usr/bin/env node
2  use strict;
3
4  function writeToConsole(message) { message = "starting"
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```

Paused on breakpoint

- Watch
- Breakpoints
- Scope
 - Local
 - `this`: undefined
 - `message`: "starting"
 - Script
 - `p`: <value unavailable>
 - Global
- Call Stack
 - `writeToConsole` index.js:5
 - `(anonymous)` index.js:8
- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners
- Event Listener Breakpoints
- CSP Violation Breakpoints

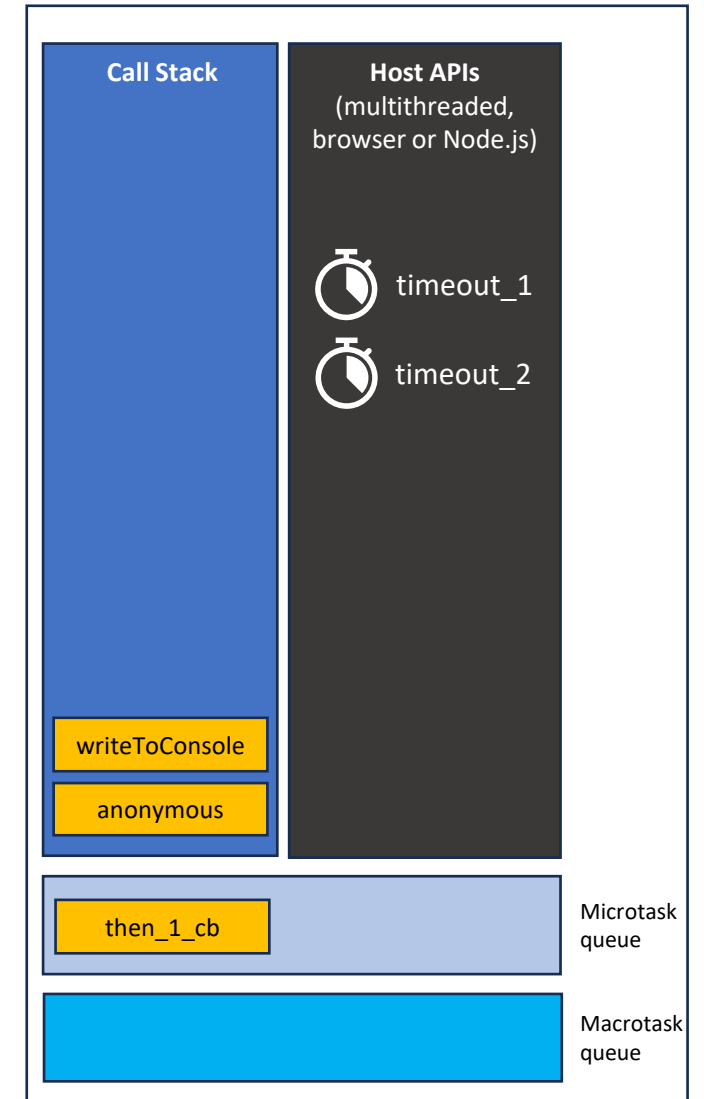


Event 1: Macrotask (cont'd)

```
1  #!/usr/bin/env node
2  use strict;
3
4  function writeToConsole(message) { message = "ending"
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```

Paused on breakpoint

- Watch
- Breakpoints
- Scope
 - Local
 - this: undefined
 - message: "ending"
 - Script
 - p: Promise {<fulfilled>: undefined}
 - Global
 - Window
- Call Stack
 - writeToConsole index.js:5
 - (anonymous) index.js:28
- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners
- Event Listener Breakpoints
- CSP Violation Breakpoints



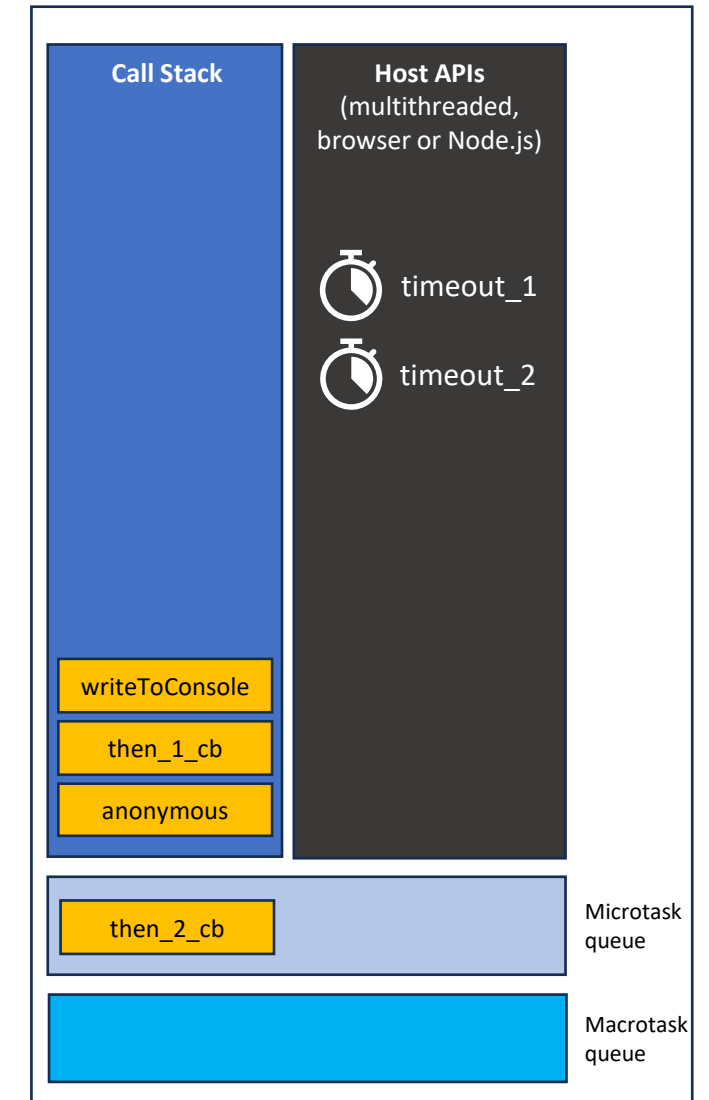
Event 2: Microtask then_1_cb

The screenshot shows a code editor with the following JavaScript code:

```
1  #!/usr/bin/env node
2  'use strict';
3
4  function writeToConsole(message) { message = "then 1"
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```

The developer tools are open, showing the program is "Paused on breakpoint" at line 5. The "Scope" panel shows the local variables `this` (undefined) and `message` ("then 1"). The "Call Stack" panel shows the following frames:

- `writeToConsole` (index.js:5)
- `then_1_cb` (index.js:23)
- `Promise.then (async)`
- `(anonymous)` (index.js:22)

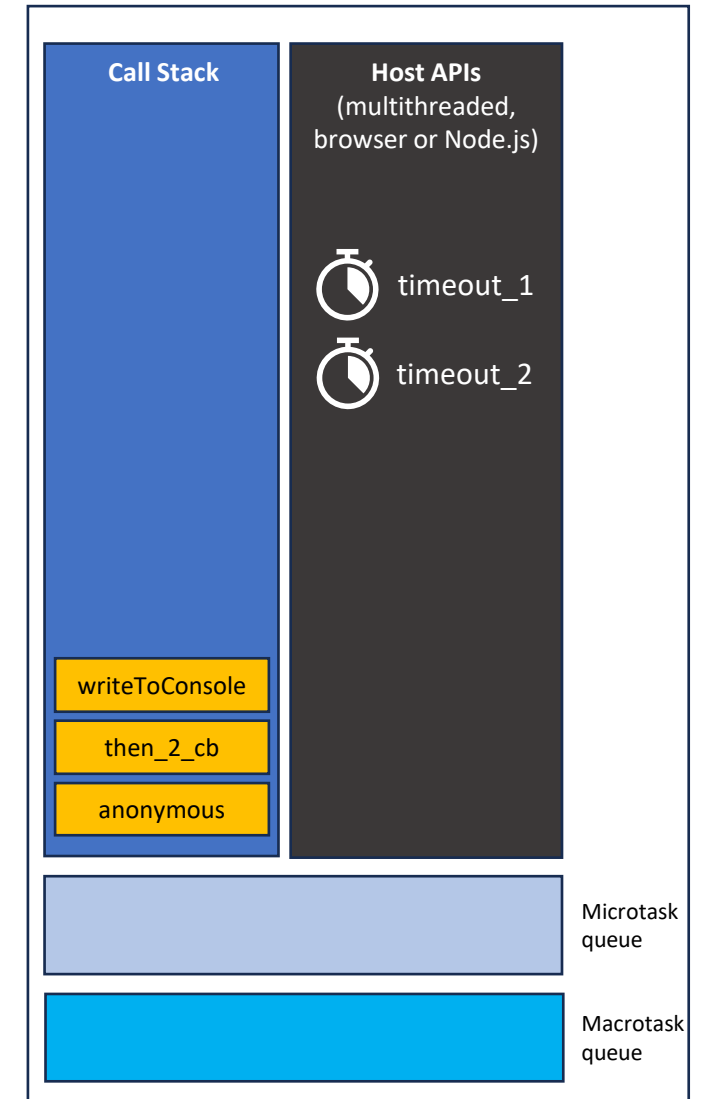


Event 3: Microtask then_2_cb

```
1  #!/usr/bin/env node
2  'use strict';
3
4  function writeToConsole(message) { message = "then 2"
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```

Paused on breakpoint

- Watch
- Breakpoints
- Scope
 - Local
 - this: undefined
 - message: "then 2"
 - Script
 - p: Promise {<fulfilled>: undefined}
 - Global
 - Window
- Call Stack
 - writeToConsole index.js:5
 - then_2_cb index.js:25
 - Promise.then (async) —
 - (anonymous) index.js:24
- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners
- Event Listener Breakpoints
- CSP Violation Breakpoints

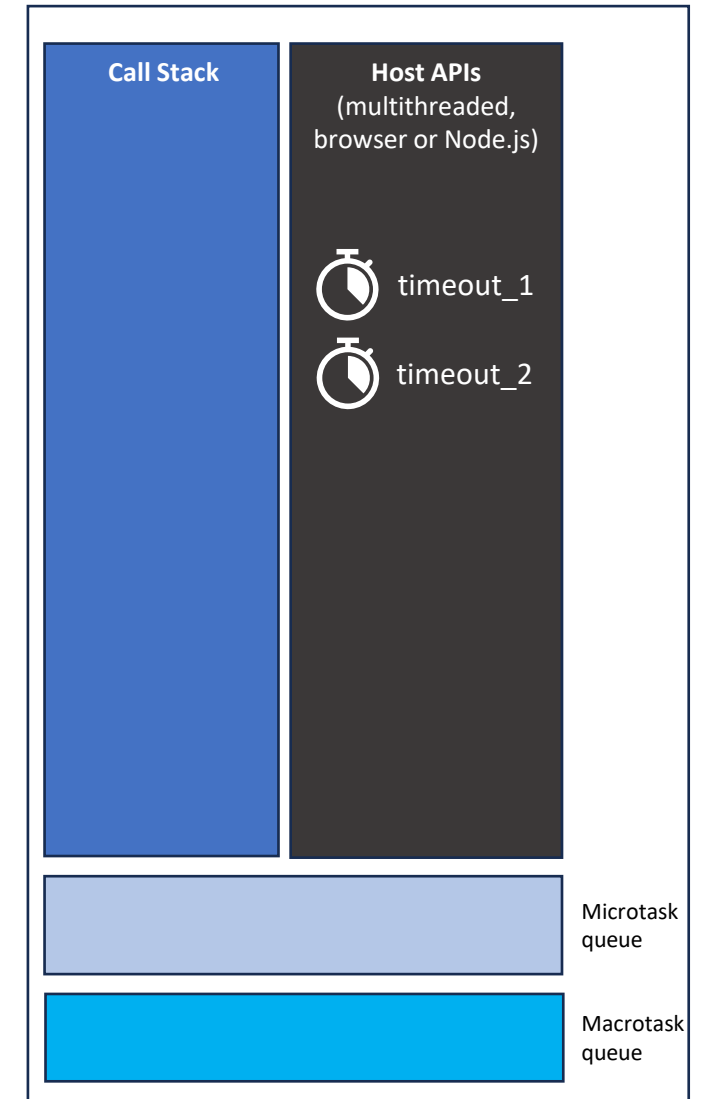


Idle, awaiting timeout_1

```
index.js X Grammarly-check.js
1  #!/ What will be printed to the console and in what order
2  'use strict';
3
4  function writeToConsole(message) {
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```

Debugger sidebar:

- Watch
- Breakpoints
- Scope
- Call Stack (Not paused)
- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners
- Event Listener Breakpoints
- CSP Violation Breakpoints

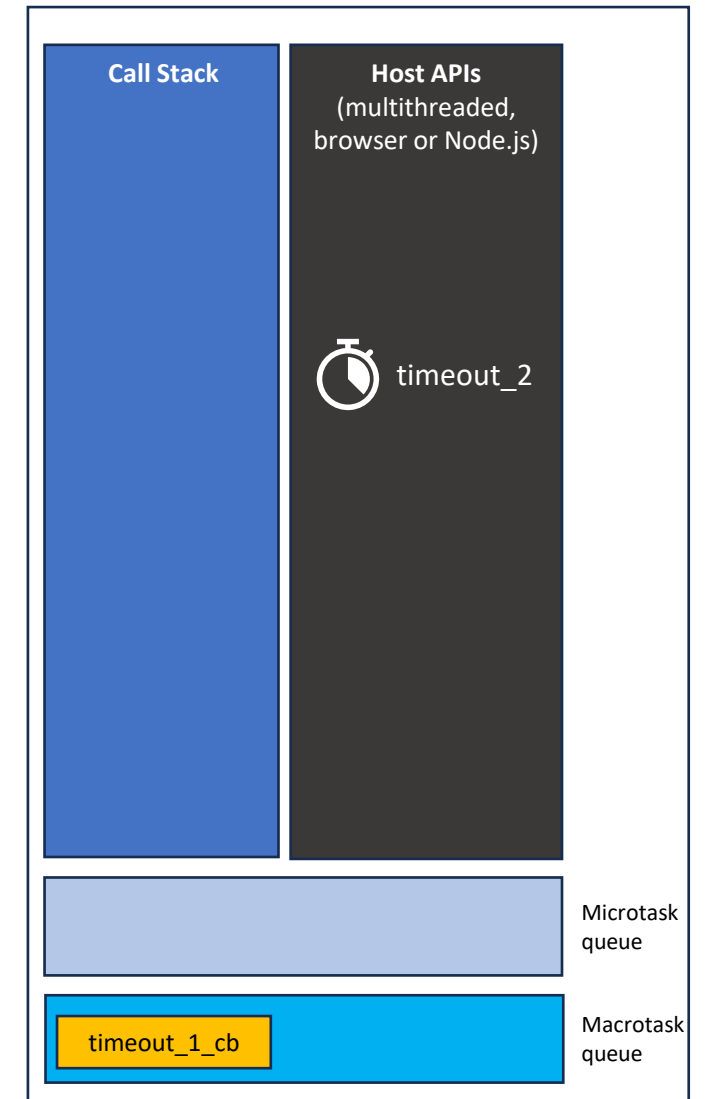


timeout_1 fired

```
index.js X Grammarly-check.js
1  ///! What will be printed to the console and in what order
2  'use strict';
3
4  function writeToConsole(message) {
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```

Debugger sidebar:

- Watch
- Breakpoints
- Scope
- Call Stack (Not paused)
- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners
- Event Listener Breakpoints
- CSP Violation Breakpoints



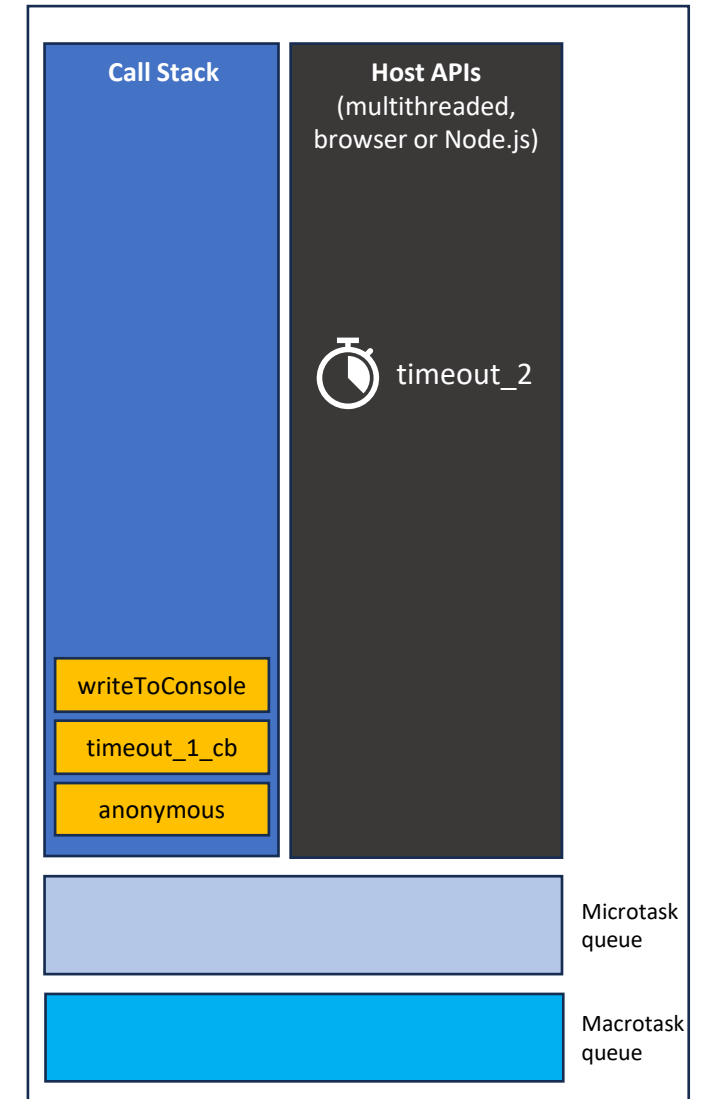
Event 4: Macrotask timeout_1_cb

The screenshot shows a code editor with the following JavaScript code in `index.js`:

```
1  #!/usr/bin/env node
2  'use strict';
3
4  function writeToConsole(message) { message = "timeout 1"
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```

The code is paused on a breakpoint at line 5, `console.log(message);`. The DevTools interface shows the following details:

- Paused on breakpoint**: The status bar indicates the execution is paused.
- Scope**: The local scope shows `this: undefined` and `message: "timeout 1"`. The script scope shows `p: Promise {<fulfilled>: undefined}`. The global scope is `Window`.
- Call Stack**: The stack is highlighted with a red box, showing the following frames:
 - `writeToConsole` at `index.js:5`
 - `timeout_1_cb` at `index.js:11`
 - `setTimeout (async)`
 - `(anonymous)` at `index.js:10`
- Breakpoints**: The left sidebar shows various breakpoint categories like XHR/fetch, DOM, Global Listeners, etc.

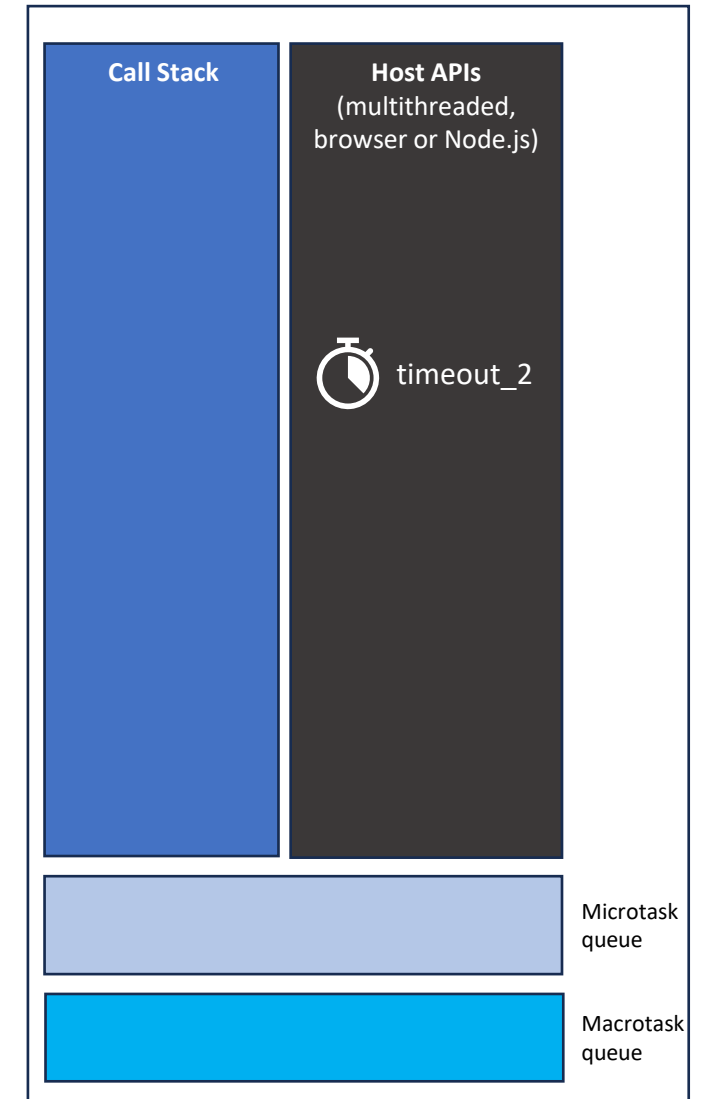


Idle 2, awaiting timeout_2

```
index.js X Grammarly-check.js
1  #!/usr/bin/env node
2  'use strict';
3
4  function writeToConsole(message) {
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```

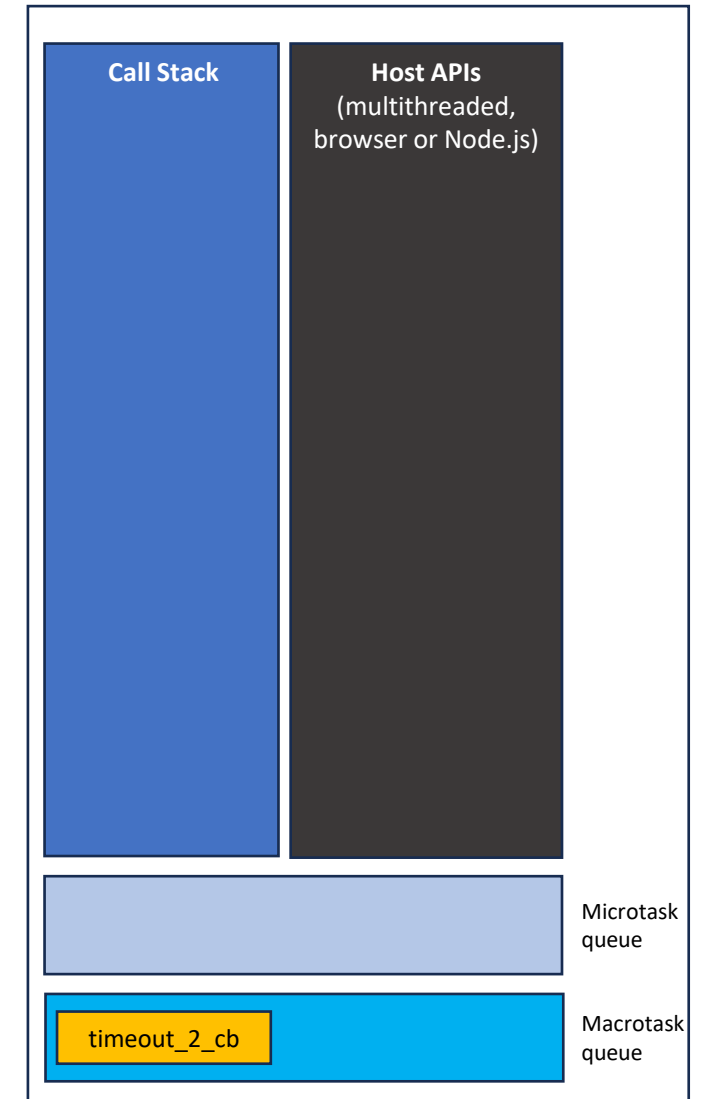
Debugger sidebar:

- Watch
- Breakpoints
- Scope
- Call Stack (Not paused)
- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners
- Event Listener Breakpoints
- CSP Violation Breakpoints



timeout_2 fired

```
1 //! What will be printed to the console and in what order
2 'use strict';
3
4 function writeToConsole(message) {
5   console.log(message);
6 }
7
8 writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```



Event 5: Macrotask timeout_2_callback

The screenshot shows a code editor with the following JavaScript code:

```
1  #!/usr/bin/env node
2  'use strict';
3
4  function writeToConsole(message) { message = "timeout 2"
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10 setTimeout(function timeout_1_cb() {
11   writeToConsole('timeout 1');
12 }, 1000);
13
14 setTimeout(function timeout_2_cb() {
15   writeToConsole('timeout 2');
16 }, 2000);
17
18 const p = new Promise((resolve, reject) => {
19   resolve();
20 });
21
22 p.then(function then_1_cb() {
23   writeToConsole('then 1');
24 }).then(function then_2_cb() {
25   writeToConsole('then 2');
26 });
27
28 writeToConsole('ending');
29
```

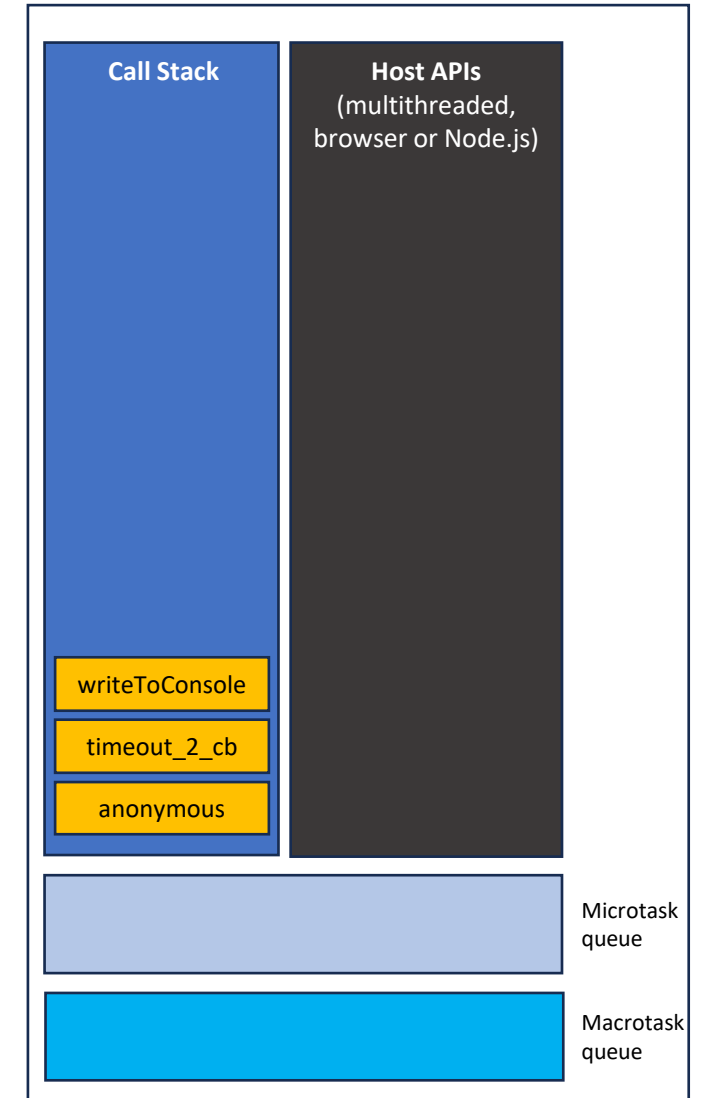
The Chrome DevTools interface is shown on the right, indicating the execution is **Paused on breakpoint** at line 5. The **Scope** panel shows the current context:

- Local
 - `this`: undefined
 - `message`: "timeout 2"
- Script
 - `p`: Promise {<fulfilled>: undefined}
- Global
 - `Window`

The **Call Stack** panel shows the following frames:

- `writeToConsole` (index.js:5) - **Current frame**
- `timeout_2_cb` (index.js:15)
- `setTimeout (async)` (index.js:14)
- `(anonymous)` (index.js:14)

Below the call stack, various breakpoint categories are listed: XHR/fetch Breakpoints, DOM Breakpoints, Global Listeners, Event Listener Breakpoints, and CSP Violation Breakpoints.



Program finished

```
1  //! What will be printed to the console and in what order
2  'use strict';
3
4  function writeToConsole(message) {
5    console.log(message);
6  }
7
8  writeToConsole('starting');
9
10  setTimeout(function timeout_1_cb() {
11    writeToConsole('timeout 1');
12  }, 1000);
13
14  setTimeout(function timeout_2_cb() {
15    writeToConsole('timeout 2');
16  }, 2000);
17
18  const p = new Promise((resolve, reject) => {
19    resolve();
20  });
21
22  p.then(function then_1_cb() {
23    writeToConsole('then 1');
24  }).then(function then_2_cb() {
25    writeToConsole('then 2');
26  });
27
28  writeToConsole('ending');
29
```

