Cooperative Scheduling

BlinkOn9

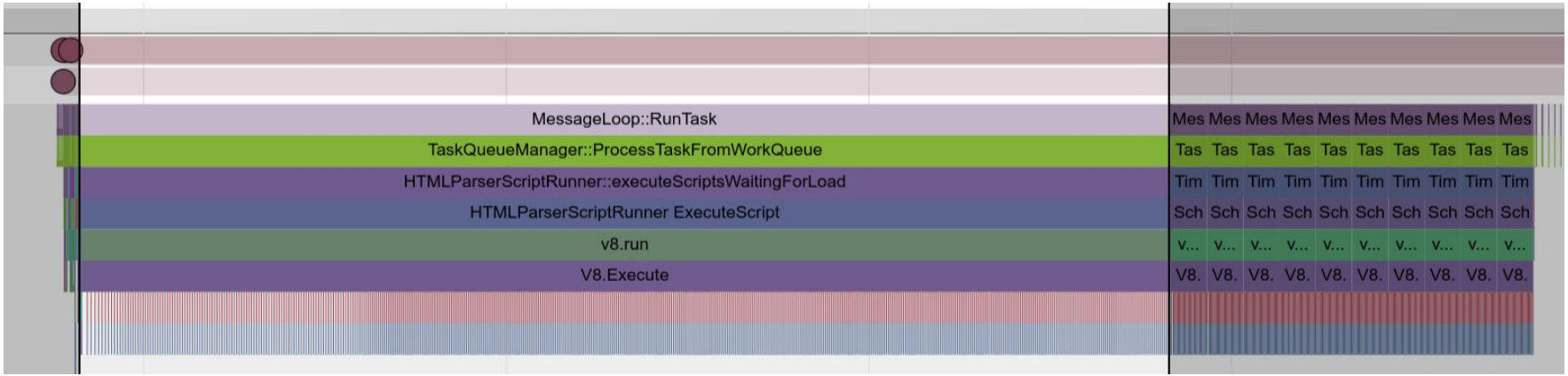
tzik@

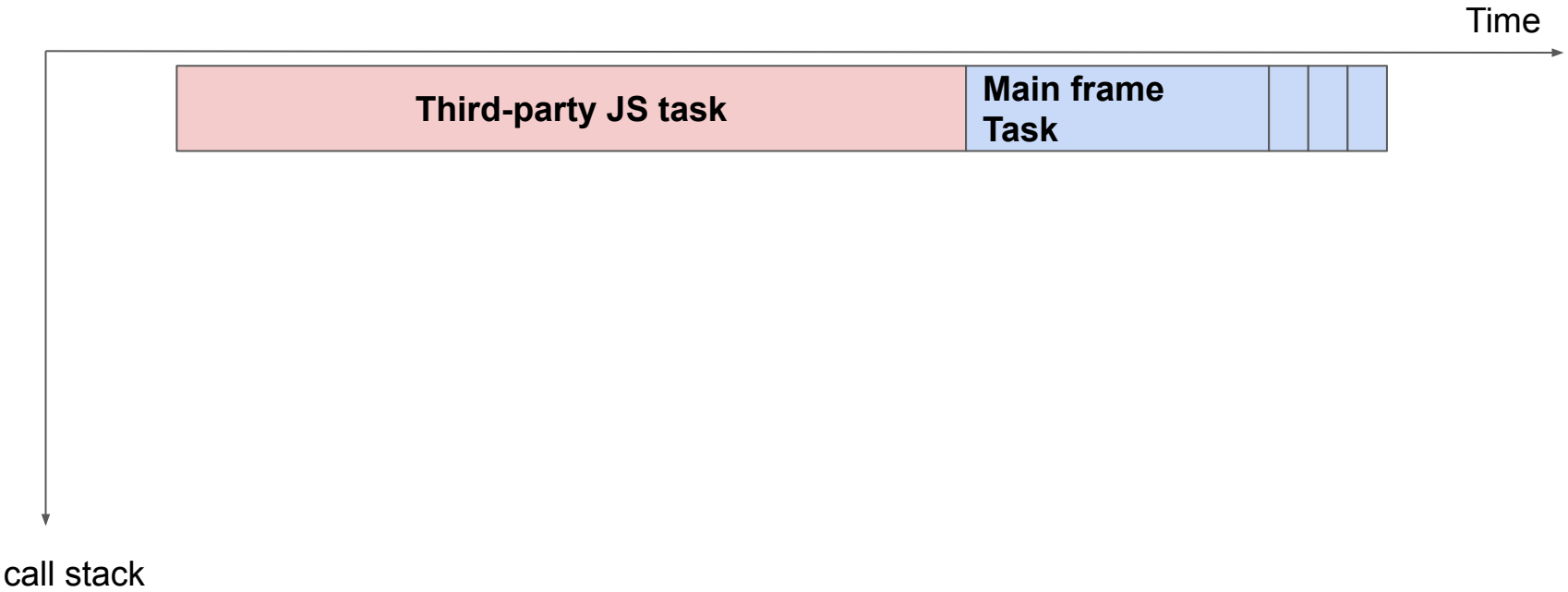
2018-04-19

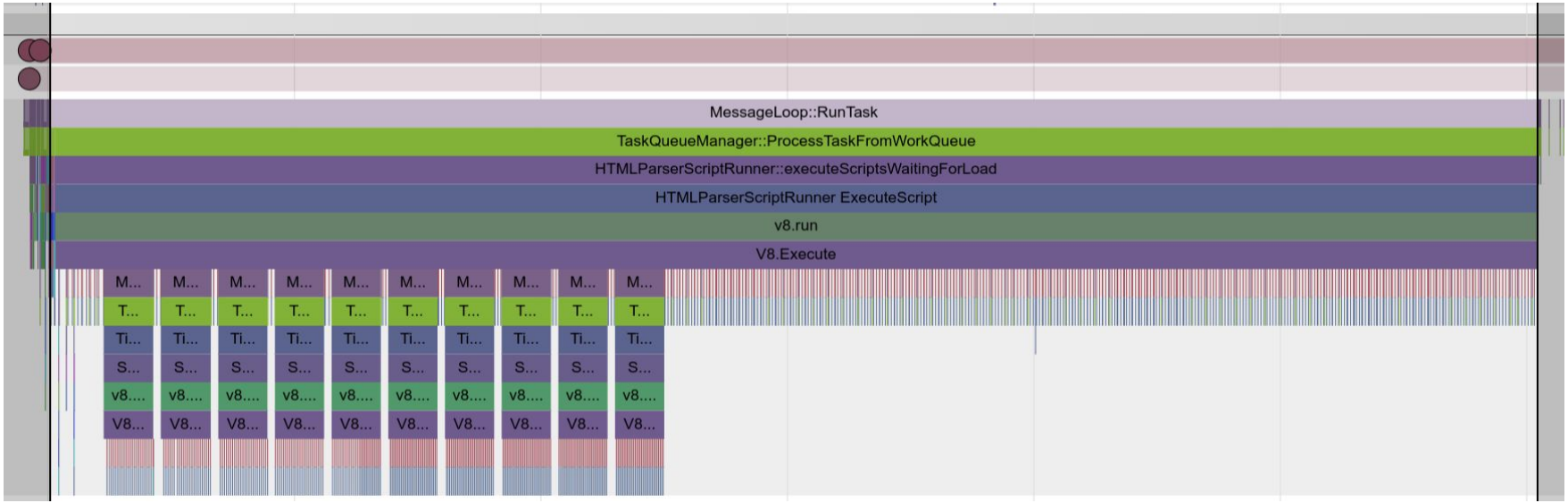
TL;DR: Cooperative Scheduling

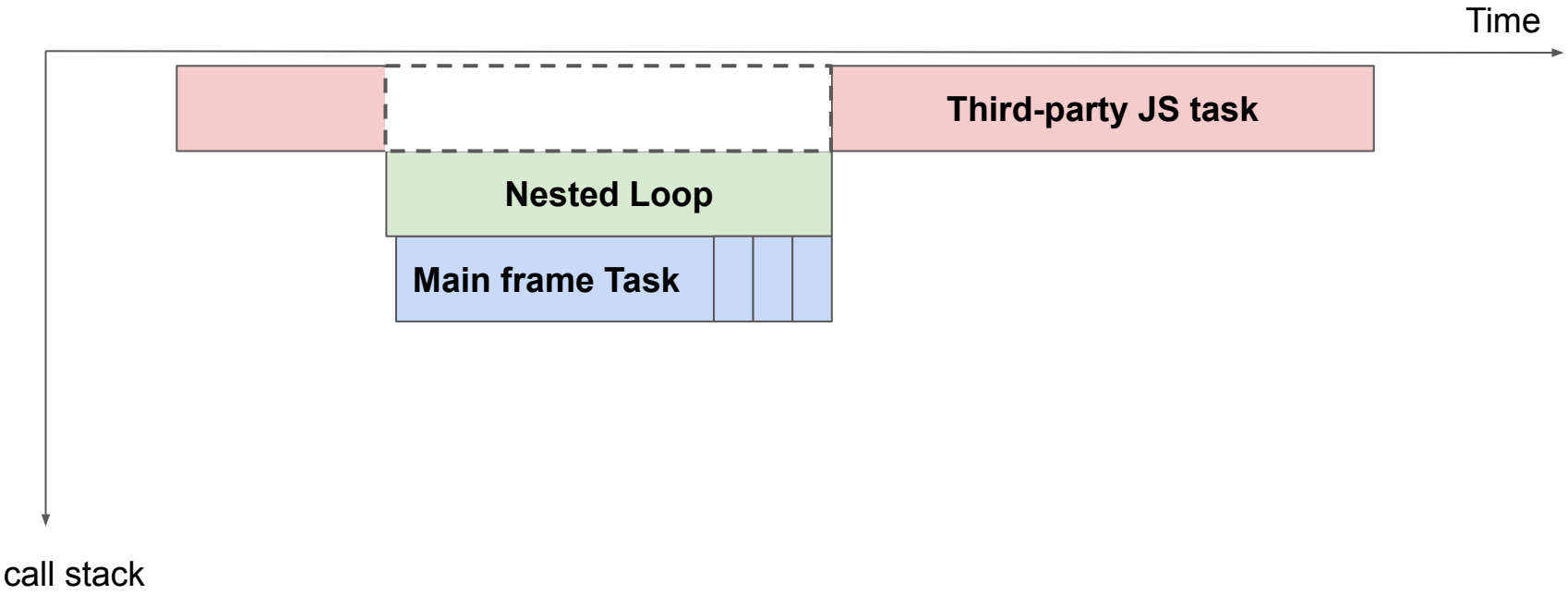
- **Pause** long-running JS task
- **Run** a Nested Loop for other tasks
- **Improve** the renderer responsiveness

Design Doc: Cooperative Scheduling


MessageLoop::RunTask
TaskQueueManager::ProcessTaskFromWorkQueue
HTMLParserScriptRunner::executeScriptsWaitingForLoad
HTMLParserScriptRunner ExecuteScript
v8.run
V8.Execute


Time
Third-party JS task
Main frame
Task
call stack


MessageLoop::RunTask
TaskQueueManager::ProcessTaskFromWorkQueue
HTMLParserScriptRunner::executeScriptsWaitingForLoad
HTMLParserScriptRunner ExecuteScript
v8.run
V8.Execute


Time
Third-party JS task
Nested Loop
Main frame Task
call stack

Impact?

As a preliminary result from a prototype impl.

# of slow input handling (>100ms input latency) is reduced by 30% by CoopSched.

CoopSched vs OOPIF

OOPIF = **O**ut **O**f **P**rocess **IF**rame

	CoopSched	OOPIF
Security	No difference	✅ Process level isolated
Responsiveness	✅ Yield main thread	✅ Free up the main thread
Memory	✅ No new process needed	Needs an extra process

CoopSched → Android

OOPIF → Others

Deep dive

Controlling Reentrancy

We **must not re-enter** to the author script, while it's **paused**.

```
let b = true;
function foo() {
  Promise.resolve().then(bar);
  b = false;
  // Checkpoint. Nested loop may run here.
  document.title;
  b = true;
}

function bar() {
  console.assert(b, "|b| should be true!!!1");
}
```

Safe to re-enter?

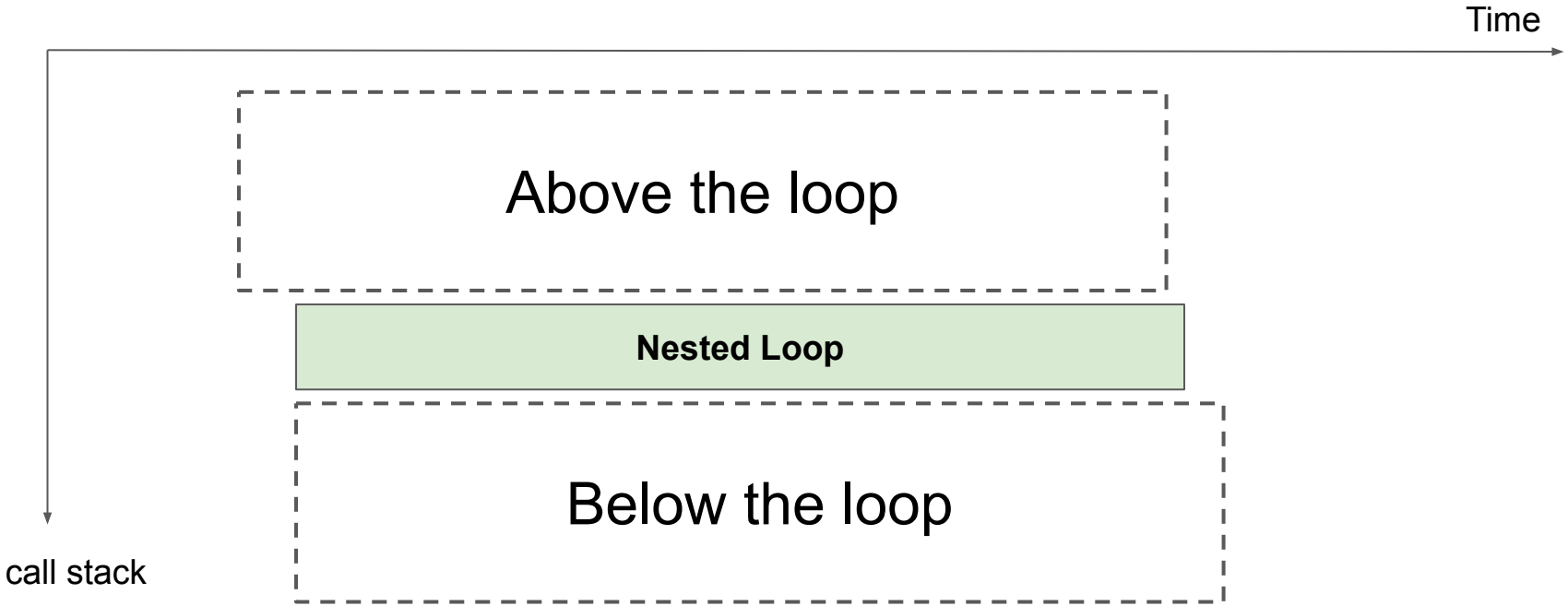
Safe

- V8 (re-entrance from callbacks)
- Blink Scheduler (from its tasks)
- Other whitelisted components
  - ScriptRunner
  - HTMLParserScriptRunner

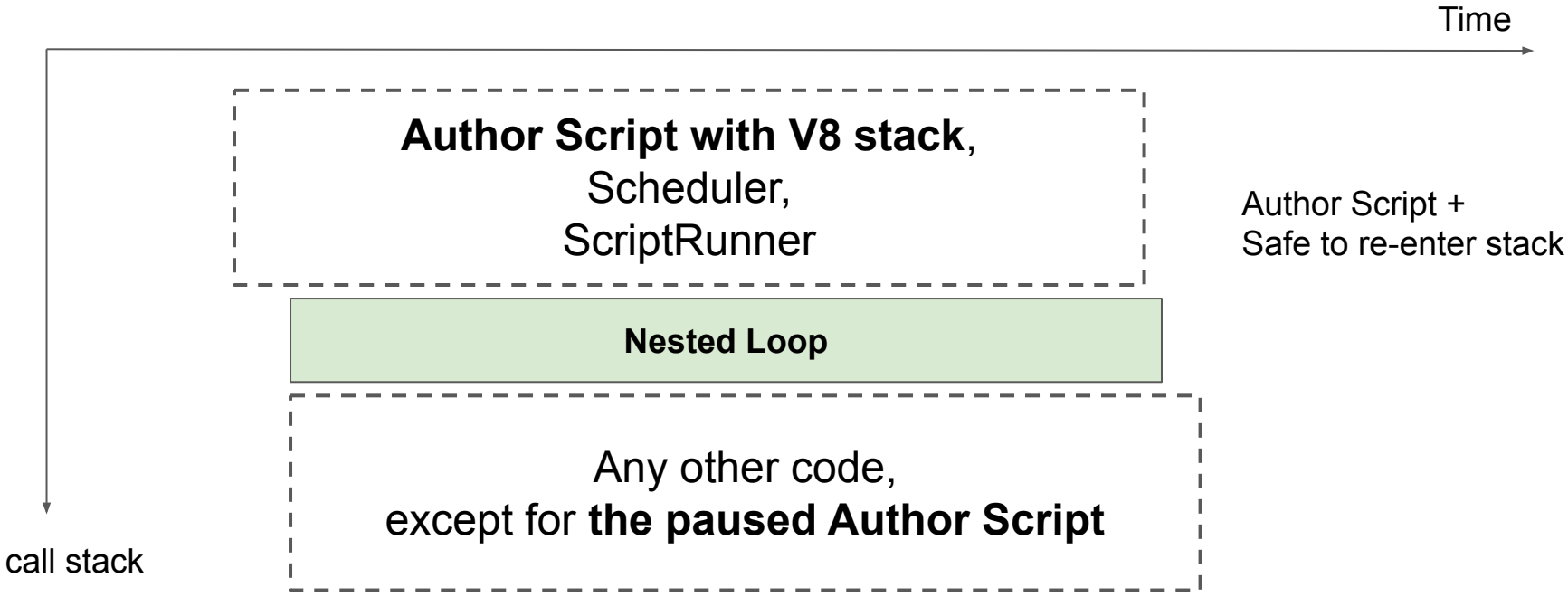
Unsafe

- **Author Script**
- **All other Blink code**

Controlling Reentrancy



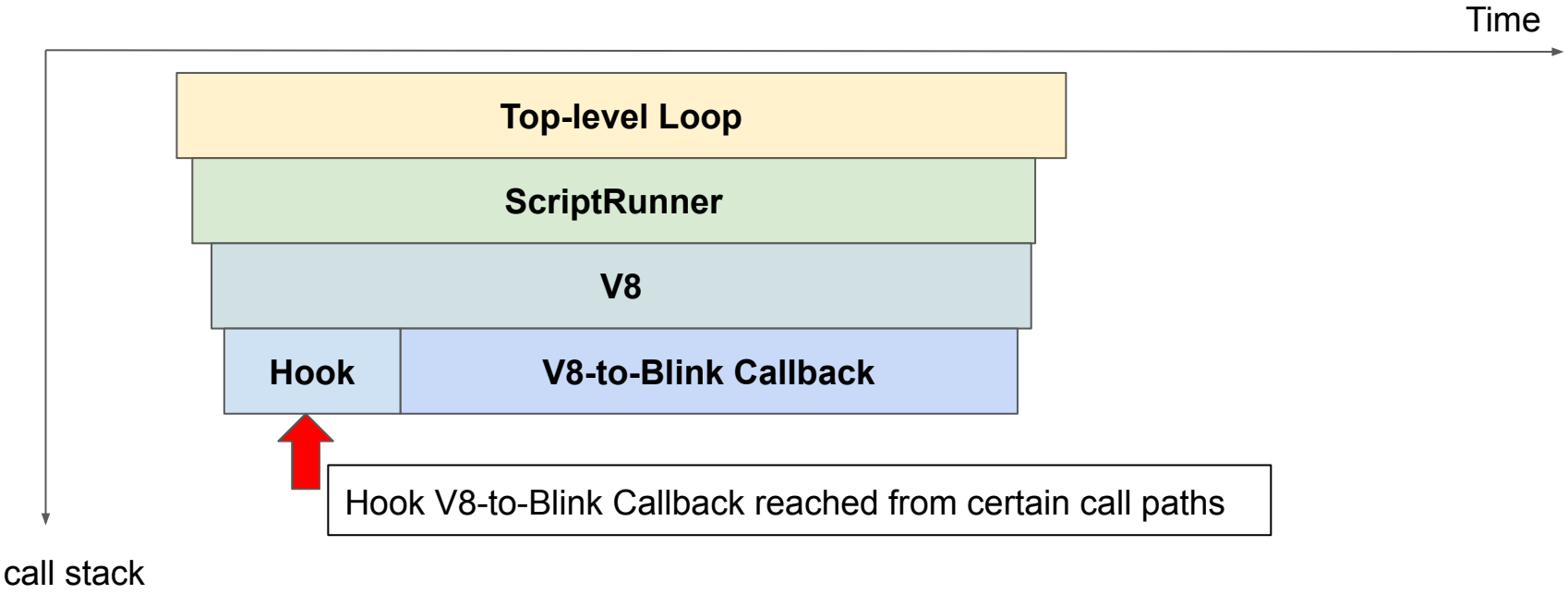
Controlling Reentrancy



Controlling Reentrancy

- Run the Nested Loop only when **above the loop stack contains known to be re-enter safe items, plus an author script to be paused**.
  - Introduce an opt-in scope to CoopSched to mark the stack is clean.
  - Hook V8-to-Blink callbacks as the nested loop safepoints.
- And, keep that **below the loop stack don’t run the paused author script** directly or indirectly.
  - Stop task scheduler for the paused frames
  - Stop running V8 microtasks for the paused frames
  - Use RemoteFrame to avoid other synchronous events

Maintaining “Above the Loop” stack



Maintaining “Below the Loop” stack

Run anything but **paused author script.**

- Pause Task Scheduler for the paused frame.
  - Pause the same eTLD+1 frames as well.
- Pause V8 MicrotaskQueue for the paused frame
  - Requires MicrotaskQueue split
- Convert synchronous events to asynchronous
  - or just use RemoteFrame everywhare

Synchronous cross-frame access

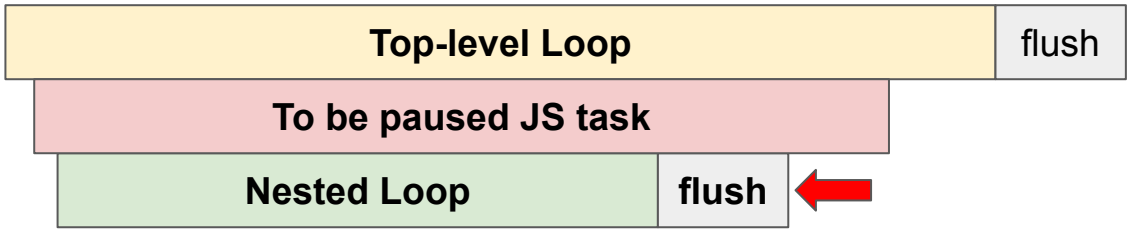
- Same origin-domain frames can access each other synchronously.
- Frames can be same origin-domain if their schemes and eTLD+1 are same.
  - e.g: https://foo.example.com and https://bar.example.com:8888 can be same origin-domain, if document.domain = “example.com” is done in both frame.

```
<iframe id="foo" src="foo.html"></iframe>
<script>
  document.getElementById("foo").contentWindow.bar();
</script>
```

→ Pause the same eTLD+1 frames as well.

Microtask Queue

- Microtask Queue is mainly for running Promise handler.
- Flushed for each end of the message loop.
- One queue per thread for now.



- The nested loop needs to flush all microtasks but paused ones.

→ Split Microtask Queue for each group of the same eTLD+1 frames

</slide>

wip/backup slides below

Avoid synchronous events

Some events are dispatched synchronously to the script. E.g. onbeforeunload and onunload. We need to convert them to asynchronous, or use RemoteFrame even for same process third party iframes.

Pausing the default task runner?

Most of tasks that run JS are moved to per-frame task runner. And there is a plan to ban JS from the default task runner.

→ Pause default task runner as well while running nested loop, until we ban JS from the default task runner.