# Requesting performance isolation with the Origin-Agent-Cluster header


- Browser compatibility
- Why browsers can't automatically segregate same-site origins
- What origin-keyed pages cannot do {: #limitations }
- When to use origin-keyed agent clusters
  - How is this related to cross-origin isolation? {: #cross-origin-isolation }
- How to use the Origin-Agent-Cluster header {: #how-to-use }
- Origin-keying is not a security feature
- Feedback
- Learn more


`Origin-Agent-Cluster` is a new HTTP response header that instructs the browser to prevent synchronous scripting access between same-site cross-origin pages. Browsers may also use `Origin-Agent-Cluster` as a hint that your origin should get its own, separate resources, such as a dedicated process.

## Browser compatibility

Currently the `Origin-Agent-Cluster` header is only implemented in Chrome 88 onward. It was designed in close collaboration with representatives from Mozilla Firefox who have marked it as worth prototyping, and has a preliminary positive reception from representatives of WebKit, the browser engine used by Safari.

But in the meantime, there's no problem with deploying the `Origin-Agent-Cluster` header to all your users today. Browsers which don't understand it will just ignore it. And, since pages in origin-

keyed agent clusters can actually do fewer things than site-keyed ones (the default), there's no interoperability issue to be worried about.

# Why browsers can't automatically segregate same-site origins

The web is built on the same-origin policy, which is a security feature that restricts how documents and scripts can interact with resources from another origin. For example, a page hosted at `https://a.example` is at a different origin from one at `https://b.example`, or one at `https://sub.a.example`.

Behind the scenes, browsers use the separation that origins provide in different ways. In the old days, even though separate origins would not be able to access each other's data, they would still share resources like operating system threads, processes, and memory allocation. This meant that if one tab was slow, it would slow down all the other tabs. Or if one tab used too much memory, it would crash the entire browser.

These days browsers are more sophisticated, and try to separate different origins into different processes. How exactly this works varies per browser: most browsers have some level of separation between tabs, but different iframes inside a single tab might share a process. And because processes come with some memory overhead, they use heuristics to avoid spawning too many: for example, Firefox has a user-configurable process limit, and Chrome varies its behavior between desktop (where memory is more plentiful) and mobile (where it is scarce).

These heuristics are not perfect. And they suffer from an important limitation: because there are exceptions to the same-origin policy which allow subdomains like `https://sub.a.example` and `https://a.example` to talk to each other, browsers cannot automatically segregate subdomains from each other.

The technical distinction here is that the browser cannot automatically segregate pages which are same-site to each other, even if they are cross-origin. The most common cases of same-site cross-origin pages happen with subdomains, but see the article Understanding "same-site" and "same-origin" for more.

This default behavior is called "site-keyed agent clusters": that is, the browser groups pages based on their *site*. The new `Origin-Agent-Cluster` header asks the browser to change this default behavior for a given page, putting it into an *origin*-keyed agent cluster, so that it is grouped only with other pages that have the exact same origin. In particular, same-site cross-origin pages will be excluded from the agent cluster.

This opt-in separation allows browsers to give these new origin-keyed agent clusters their own dedicated resources, which are not combined with those of other origins. For example, such pages could get their own process, or be scheduled on separate threads. By adding the `Origin-Agent-Cluster` header to your page, you're indicating to the browser that the page would benefit from such dedicated resources.

However, in order to perform the separation, and get these benefits, the browser needs to disable some legacy features.

# What origin-keyed pages cannot do {: #limitations }

When your page is in an origin-keyed agent cluster, you give up some abilities to talk to same-site cross-origin pages that were previously available. In particular:

- You can no longer set `document.domain`. This is a legacy feature that normally allows same-site cross-origin pages to synchronously access each other's DOM, but in origin-keyed agent clusters, it is disabled.
- You can no longer send `WebAssembly.Module` objects to other same-site cross-origin pages via `postMessage()`.
- (Chrome-only) You can no longer send `SharedArrayBuffer` or `WebAssembly.Memory` objects to other same-site cross-origin pages.

Chrome is the only browser that allows sending `SharedArrayBuffer` and `WebAssembly.Memory` objects to same-site cross-origin pages. Other browsers, and future versions of Chrome, will prevent sending these objects across the origin boundary regardless of whether the agent cluster is origin-keyed or site-keyed.

# When to use origin-keyed agent clusters

The origins that most benefit from the `Origin-Agent-Cluster` header are those that:

- Perform best with their own dedicated resources when possible. Examples include performance-intensive games, video conferencing sites, or multimedia creation apps.
- Contains resource-intensive iframes that are different-origin, but same-site. For example, if `https://mail.example.com` embeds `https://chat.example.com` iframes, origin-keying `https://mail.example.com/` ensures that the code written by the chat team cannot

accidentally interfere with code written by the mail team, and can hint to the browser to give them separate processes to schedule them independently and decrease their performance impact on each other.
- Expect to be embedded on different-origin, same-site pages, but know themselves to be resource-intensive. For example, if `https://customerservicewidget.example.com` expects to use lots of resources for video chat, and will be embedded on various origins throughout `https://*.example.com`, the team maintaining that widget could use the `Origin-Agent-Cluster` header to try to decrease their performance impact on embedders.

Additionally, you'll also need to make sure you're OK disabling the above-discussed rarely-used cross-origin communication features, and that your site is using HTTPS.

But in the end, these are just guidelines. Whether origin-keyed agent clusters will help your site or not is ultimately best determined through measurements. In particular, you'll want to measure your Web Vitals, and potentially your memory usage, to see what impact origin-keying has. (Memory usage in particular is a potential concern, as increasing the number of processes in play can cause more per-process memory overhead.) You shouldn't just roll out origin-keying and hope for the best.

# How is this related to cross-origin isolation? {: #cross-origin-isolation }

Origin-keying of agent clusters via the `Origin-Agent-Cluster` header is related to, but separate from, cross-origin isolation via the `Cross-Origin-Opener-Policy` and `Cross-Origin-Embedder-Policy` headers.

Any site which makes itself cross-origin isolated will also disable the same same-site cross-origin communications features as when using the `Origin-Agent-Cluster` header. However, the `Origin-Agent-Cluster` header can still be useful on top of cross-origin isolation, as an additional hint to the browser to modify its resource allocation heuristics. So you should still consider applying the `Origin-Agent-Cluster` header, and measuring the results, even on pages that are already cross-origin isolated.

# How to use the `Origin-Agent-Cluster` header {: #how-to-use }

To use the `Origin-Agent-Cluster` header, configure your web server to send the following HTTP response header:

```
Origin-Agent-Cluster: ?1
```

The value of `?1` is the structured header syntax for a boolean `true` value.

It's important to send this header on *all* responses from your origin, not just some pages. Otherwise, you can get inconsistent results, where the browser "remembers" seeing an origin-keying request and so it origin-keys even on pages that don't ask for it. Or the reverse: if the first page a user visits doesn't have the header, then the browser will remember that your origin does not want to be origin-keyed, and will ignore the header on subsequent pages.

Don't forget to send the header on error pages, like your 404 page!

Why can't the browser always respect the header?

The reason for this "memory" is to ensure consistency of keying for an origin. If some pages on an origin were origin-keyed, while others weren't, then you could have two same-origin pages which were put into different agent clusters, and thus weren't allowed to talk to each other. This would be very strange, both for web developers and for the internals of the browser. So, the specification for `Origin-Agent-Cluster` instead ignores the header if it's inconsistent with what it's previously seen for a given origin. In Chrome, this will result in a console warning.

This consistency is scoped to a browsing context group, which is a group of tabs, windows, or iframes which can all reach each other via mechanisms like `window.opener`, `frames[0]`, or `window.parent`. This means that, once an origin's origin- or site-keying has been settled (by the browser either seeing, or not seeing, the header), changing it requires opening up an entirely new tab, not connected to the old one in any way.

These details can be important for testing the `Origin-Agent-Cluster` header. When first adding it to your site, just reloading the page will not work; you'll need to close the tab and open a new one.

To check whether the `Origin-Agent-Cluster` header is applied, use the JavaScript `window.originAgentCluster` property. This will be `true` in cases where the header (or other mechanisms, like cross-origin isolation) caused origin-keying; `false` when it did not; and `undefined` in browsers that don't implement the `Origin-Agent-Cluster` header. Logging this data to your analytics platform can provide a valuable check that you've configured your server correctly.

Finally, note that the `Origin-Agent-Cluster` header will only work on secure contexts, i.e. on HTTPS pages or on `http://localhost`. Non-localhost HTTP pages do *not* support origin-keyed agent clusters.

# Origin-keying is not a security feature

While using an origin-keyed agent cluster does isolate your origin from synchronous access from same-site cross-origin pages, it does not give the protection of security-related headers like `Cross-Origin-Resource-Policy` and `Cross-Origin-Opener-Policy`. In particular, it is not a reliable protection against side channel attacks like Spectre.

This might be a bit surprising, because origin-keying can sometimes cause your origin to get its own process, and separate processes are an important defence against side-channel attacks. But remember that the `Origin-Agent-Cluster` header is only a hint in that regard. The browser is under no obligation to give your origin a separate process, and it might not do so for a variety of reasons:

- A browser might not implement the technology to do so. For example, currently Safari and Firefox can put separate tabs into their own processes, but cannot yet do so for iframes.
- The browser might decide it's not worth the overhead of a separate process. For example, on low-memory Android devices, or in Android WebView, Chrome uses as few processes as possible.
- The browser might want to respect the request that the `Origin-Agent-Cluster` header indicates, but it could do so using different isolation technology than processes. For example, Chrome is exploring using threads instead of processes for this sort of performance isolation.
- The user, or code running on a different site, might have already navigated to a site-keyed page on your origin, causing the consistency guarantee to kick in and the `Origin-Agent-Cluster` header to be ignored entirely.

For these reasons, it's important not to think of origin-keyed agent clusters as a security feature. Instead, it's a way of helping the browser prioritize resource allocation, by hinting that your origin would benefit from dedicated resources (and that you're willing to give up certain features in exchange).

# Feedback

The Chrome team would love to hear from you if you're using, or considering using, the `Origin-Agent-Cluster` header. Your public interest and support helps us prioritize features and show other browser vendors how important they are. Tweet at @ChromiumDev and let Chrome DevRel know your thoughts and experiences.

If you have more questions about the specification, or the details of how the feature works, you can file an issue on the HTML Standard GitHub repository. And if you encounter any issues with Chrome's implementation, you can file a bug at new.crbug.com with the Components field set to `Internals>Sandbox>SiteIsolation`.

# Learn more

To learn more about origin-keyed agent clusters, you can dive into the details at these links:

- Demo and demo source
- Explainer
- Specification
- Tracking bugs: Chrome, Firefox, Safari

---

□□□ #3
□ □□□ □□□ 2026-10-31 22:08:19 UTC
□ cocobook-sync □□□ 2026-11-04 17:37:43 UTC