Applying CPU Prefetching Techniques to CDNs

Evan Kirkiles, B.S. Computer Science, Scott Petersen, Computer Science, Yale University

Background

Inside the CPU, all accesses to main memory first pass through a series of caches. These caches serve to vastly mitigate the high latency of main memory reads by copying data more closely to the processor. The Web is also served through a multi-level cache, but rather globally distributed, through CDNs. Caching here serves to cut out on high latency from multiple network hops and reduce duplicate work in SSG.

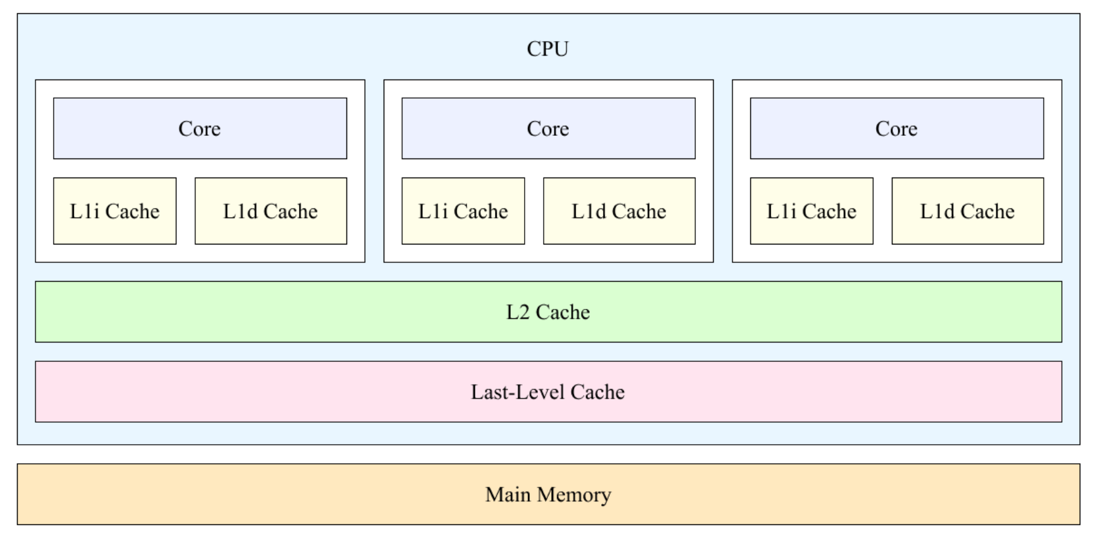

Figure 1: A simplified visualization of a three-level CPU cache hierarchy.

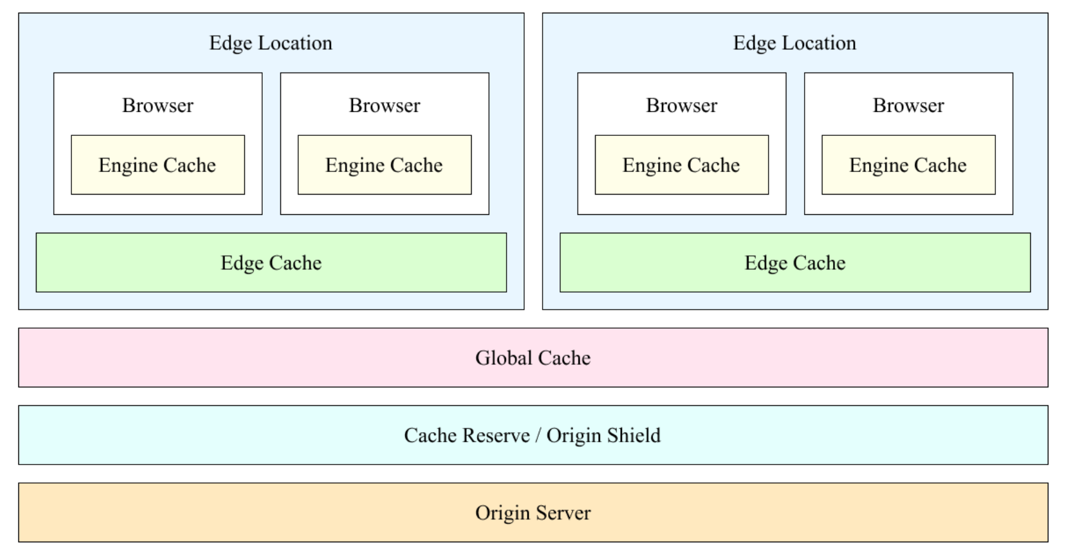

Figure 2: A simplified visualization of the caches involved in Web delivery.

Whereas the field of CPU caching is heavily studied, CDN caching is less transparent as most CDNs use proprietary implementations and hardware. The purpose of this project is to take as parallel the multi-level caches in the two systems and apply a technique from the first—instruction prefetching, important for speculative execution—and bring it to web caching as predictive prefetching.

Methods

Standard web caches populate cache entries on cache misses at request time. Our method proposes predicting user access patterns such that resources requested from a page (images, links, JavaScript, etc.) will already be present in the edge cache *before* they are requested by the user's browser—particularly on traversals of uncached pages. We compare the performance of three types of prefetch-augmented caching:

1. No Prefetching

The standard CDN behavior, in which resources missing from the cache are fetched from the origin server and populated into the cache at request time.

2. Static HTML Prefetching

As HTML resources are streamed to a requester, the `<a />` and `<img />` tags within are parsed for relative links. These links are then prefetched at the cache level, before they have reached the client (unlike the standard HTML attribute rel="prefetch").

3. Analytics-Based Prefetching

User traversals are mapped by way of the **`Referer`** HTTP header. When a page is requested, the **`n`** most frequent outgoing destinations and resources from that page are prefetched. This introduces a natural heuristic to pick the best prefetchables.

Each of the above prefetchers are implemented as TypeScript-based Cloudflare Workers, opting into and working with Cloudflare's global cache network through its Cache API.

Overview

Inspired by CPU cache behavior, we implement an augmented CDN caching layer which improves cache hit ratios—and thus, request latency—by predicting which resources will be requested after the page recently returned from cache.

Our findings show that:

- Predictive prefetching can drastically improve cache hit ratios in the context of a site exhibiting a low cache hit rate—i.e. sites without huge traffic whose content is culled from CDN edge caches frequently.
- The benefits of prefetching for sites with a high cache hit ratio are minimal or negative from the overhead of prefetch analysis on every request.
- Prefetching through static analysis, i.e. by parsing HTML, is not scaleable without changes on the side of the developers to specify prefetch resources within the DOM—something already done client-side.
- Prefetching dynamically, i.e. by analyzing user traffic, is an entirely viable way to improve early-stage edge cache hit rates while a site is not fully cached.

Results

The testing suite caches pages in a development **`wrangler`** environment from the website http://info.cern.ch. It traverses over 8 pages on the site, 3 times, with the cache purged before the final traversal, for a total of 24 requests in the testing suite.

Caching Mechanism	Average Request Latency	Cache Hit Ratio	Total Subrequests
No Prefetching	138.9ms	37.5%	54
HTML Parsing Prefetcher	62.4ms	75%	194
Analytics Prefetcher	91.1ms	62.5%	66

Table 1: Summary statistics across the experiments with different types of caching strategies. The traversal took place over 8 site pages and was repeated three times for a total of 24 navigations, with the last traversal having had the cache purged beforehand.

Importantly, the testing environment simulates a site without any cache presence—all pages (before prefetching) are cache misses until visited once. In our local environment, the latency of a cache miss is ±250ms (to fetch from the origin), whereas a cache hit is ±5ms (fetching from disk). The magnitude of the difference in these two numbers is not representative of a real world cache, though their overall difference is.

From our results:

- The HTML parsing prefetcher doubles the cache hit ratio, but quadruples the number of subrequests for blindly checking the cache status of prefetchables.
- The analytics prefetcher exhibits similar cache hit rate improvements, but for far fewer subrequests. The majority of traffic here exists in database queries.

Further research into optimal prefetching algorithms can cut the number of subrequests further—for example, weighting prefetchable resources based on their likelihood to already be cached, or moving the prefetcher to a log snooping process.

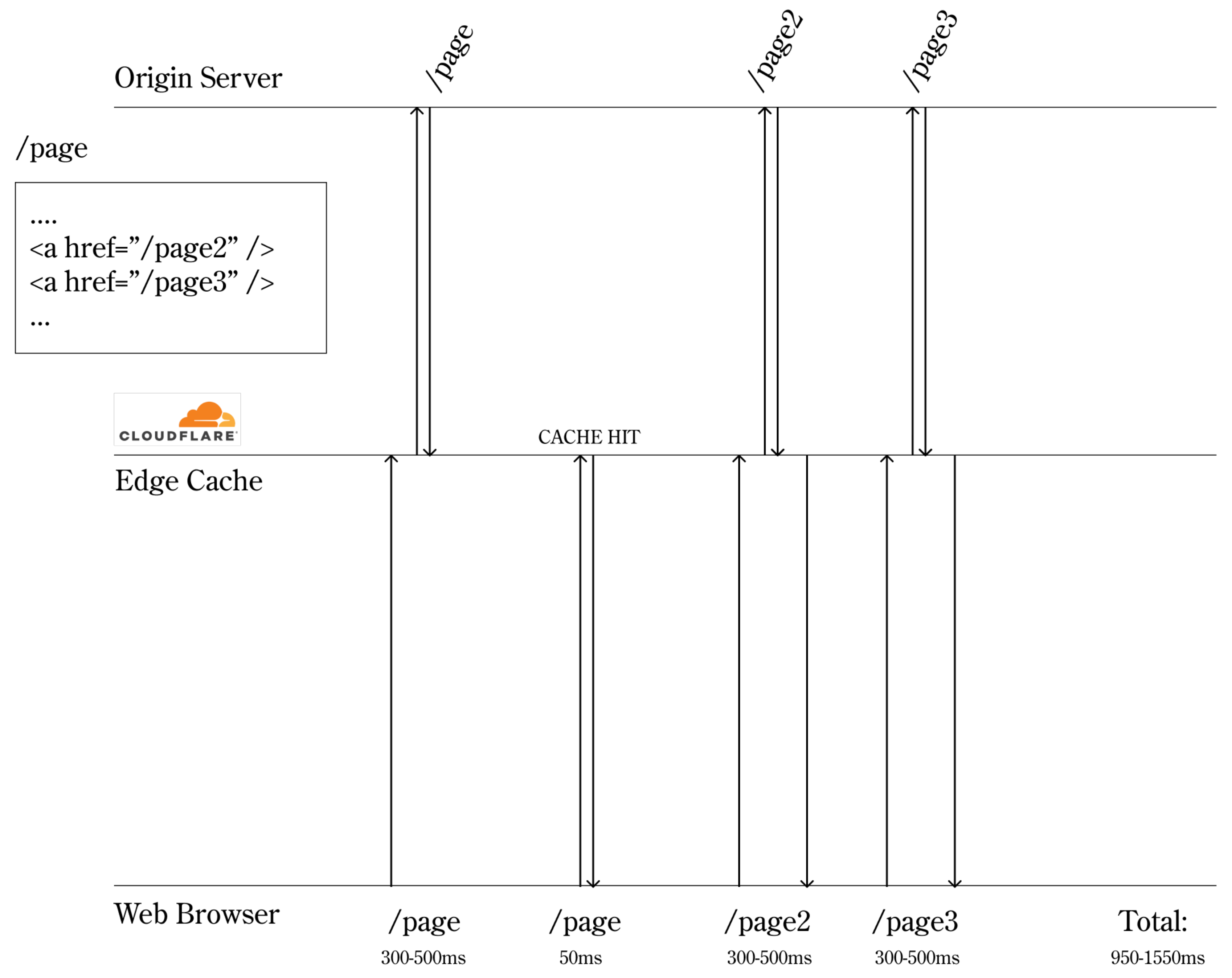

Figure 1: Fetching an example page "/page" without predictive prefetching, assuming no prior cached resources.

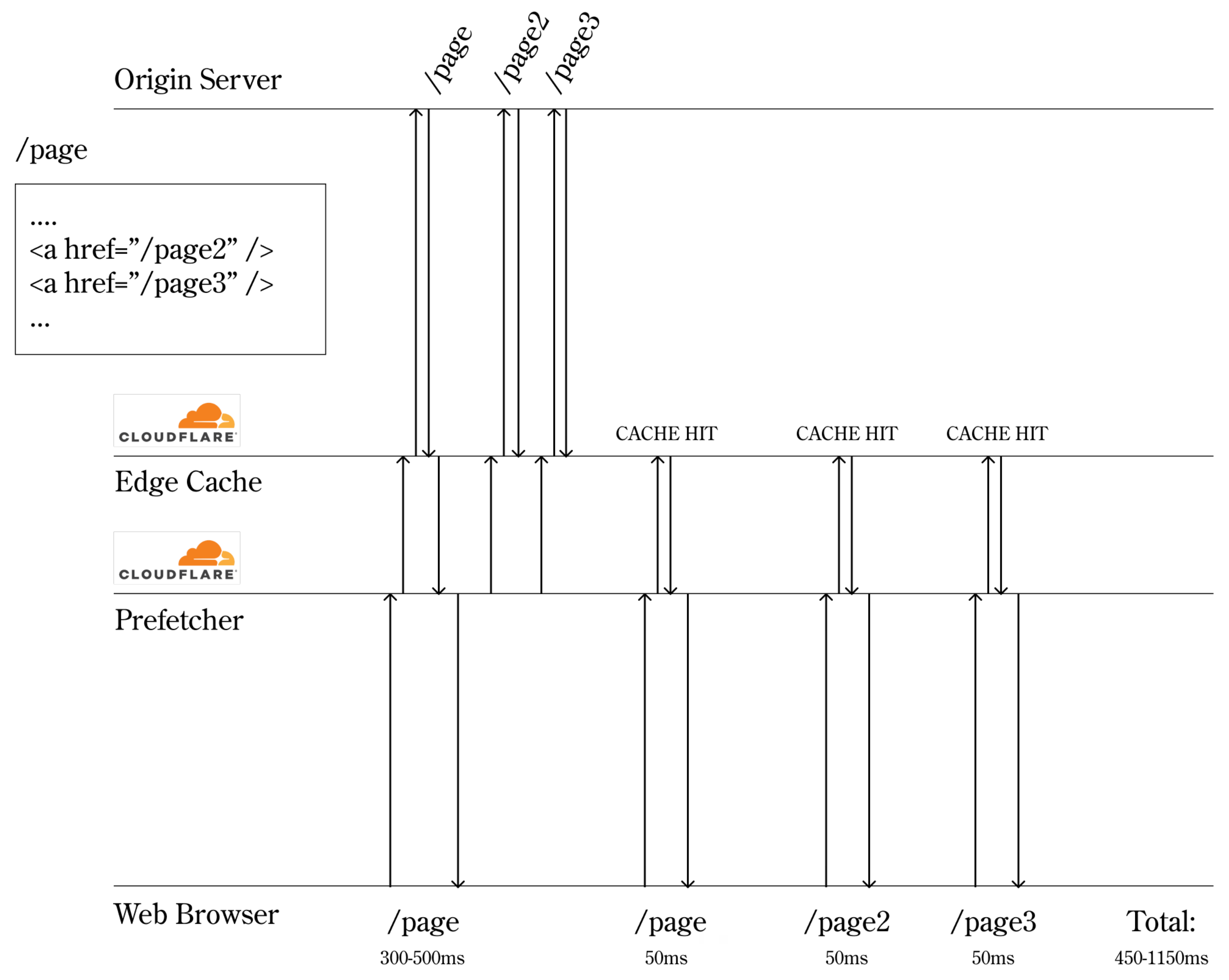

Figure 2: Fetching an example page "/page" with predictive prefetching, assuming no prior cached resources.