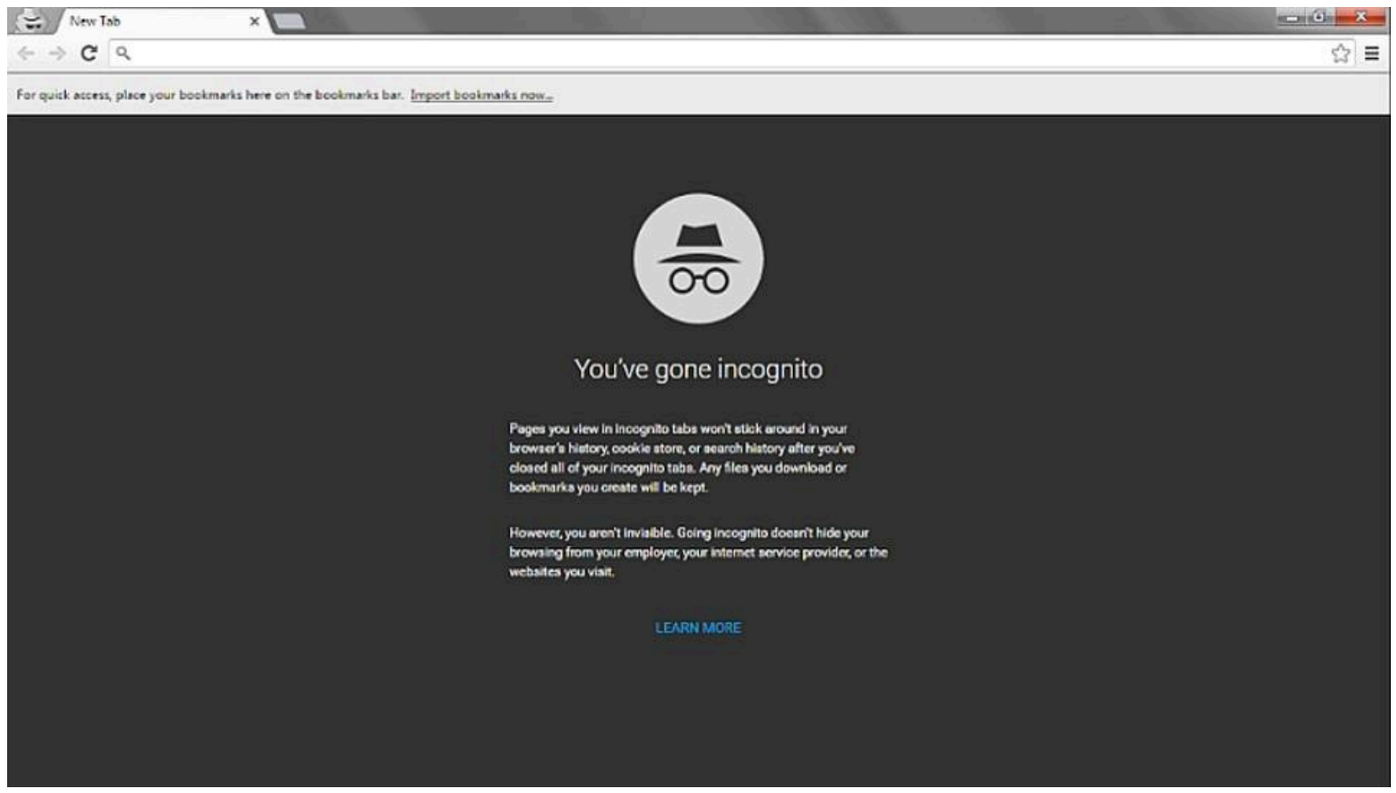
Veil: Private Browsing Semantics without Browser-side Assistance

Frank Wang (MIT CSAIL), James Mickens (Harvard),
Nickolai Zeldovich (MIT CSAIL)

All popular browsers offer private browsing



In private browsing, web pages shouldn't leave identifiable, persistent client-side state

Private browser modes are leaky

- DNS cache and database pollution
- Leave RAM artifacts in page swap, hibernation files
- Forensic tools can easily recover this data and fingerprint activity

Problem: Private browsing is hard to implement with only client-side support

- Browsers complex and constantly adding new features
- They lack a priori knowledge of sensitive content
  - Example: prevent RAM from paging to disk, use `mlock()` to pin memory
- Even transmission of web content to a user can pollute in-memory and on-disk regions

What if developers can implement private browsing semantics?

Goal: Protect greppable content from post-session attacker

Insight: Web services control

1) the content they deliver

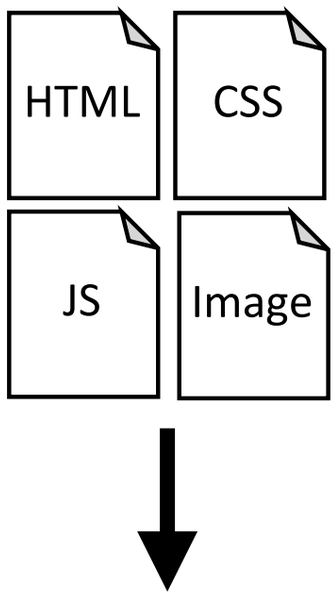
2) the servers that deliver this content

Our solution: Veil

Developer **Blinding servers** **Client browser**

Our solution: Veil

Developer **Blinding servers** **Client browser**

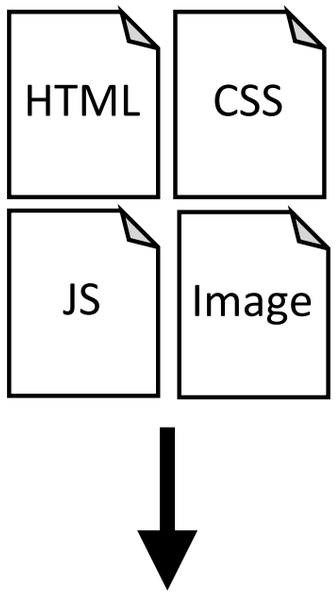


Veil Compiler

Rewrite webpages

Our solution: Veil

Developer



Veil Compiler

Rewrite webpages

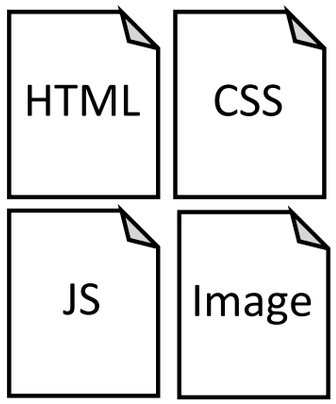
Blinding servers

Store and mutate content

Client browser

Our solution: Veil

Developer



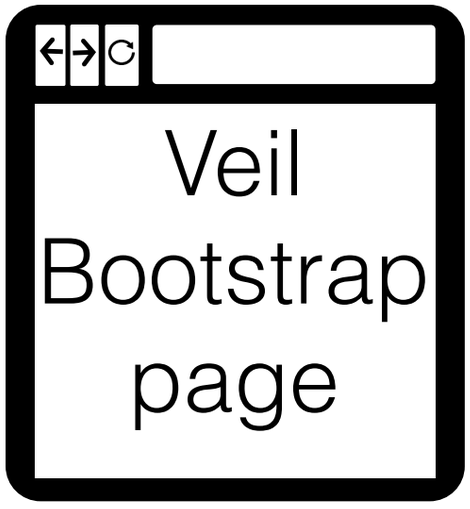
Veil Compiler

Rewrite webpages

Blinding servers

Store and mutate content

Client browser



Hide page-specific URLs and display pages

Contributions

Veil: first web framework that allows developers to provide private browsing semantics

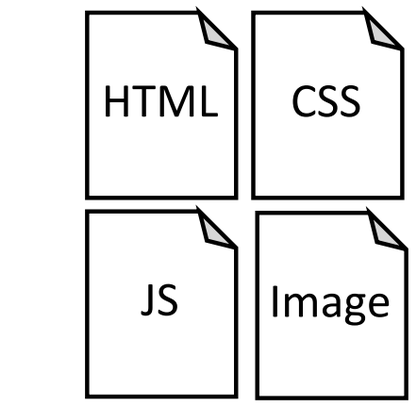
- Techniques, such as URL blinding, content mutation, and heap walking, to protect privacy
- Two browsing modes to provide different amounts of privacy
- Evaluation on real websites
- No client/browser changes required

Outline

- Veil Architecture
- Implementation
- Evaluation

Overview

Developer



Veil Compiler

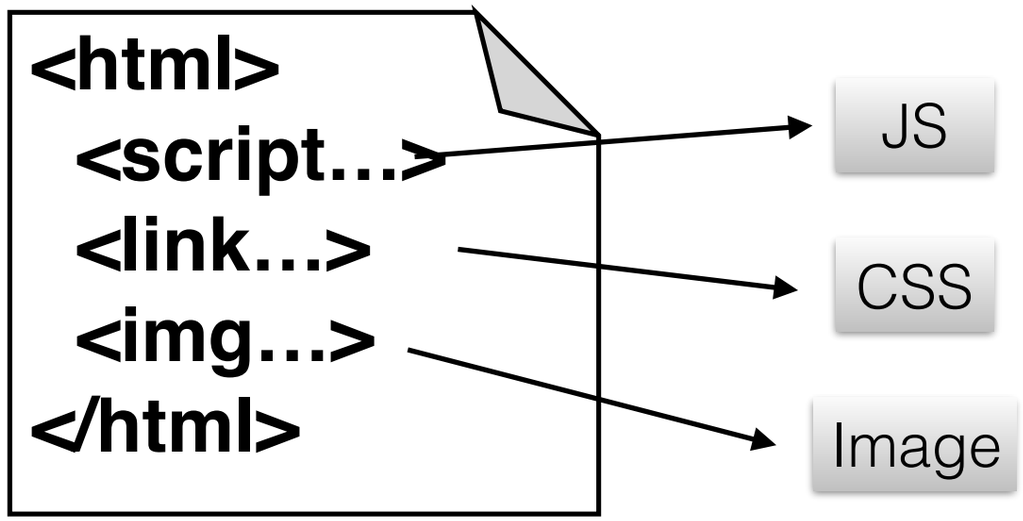
Rewritten HTML

Blinding servers

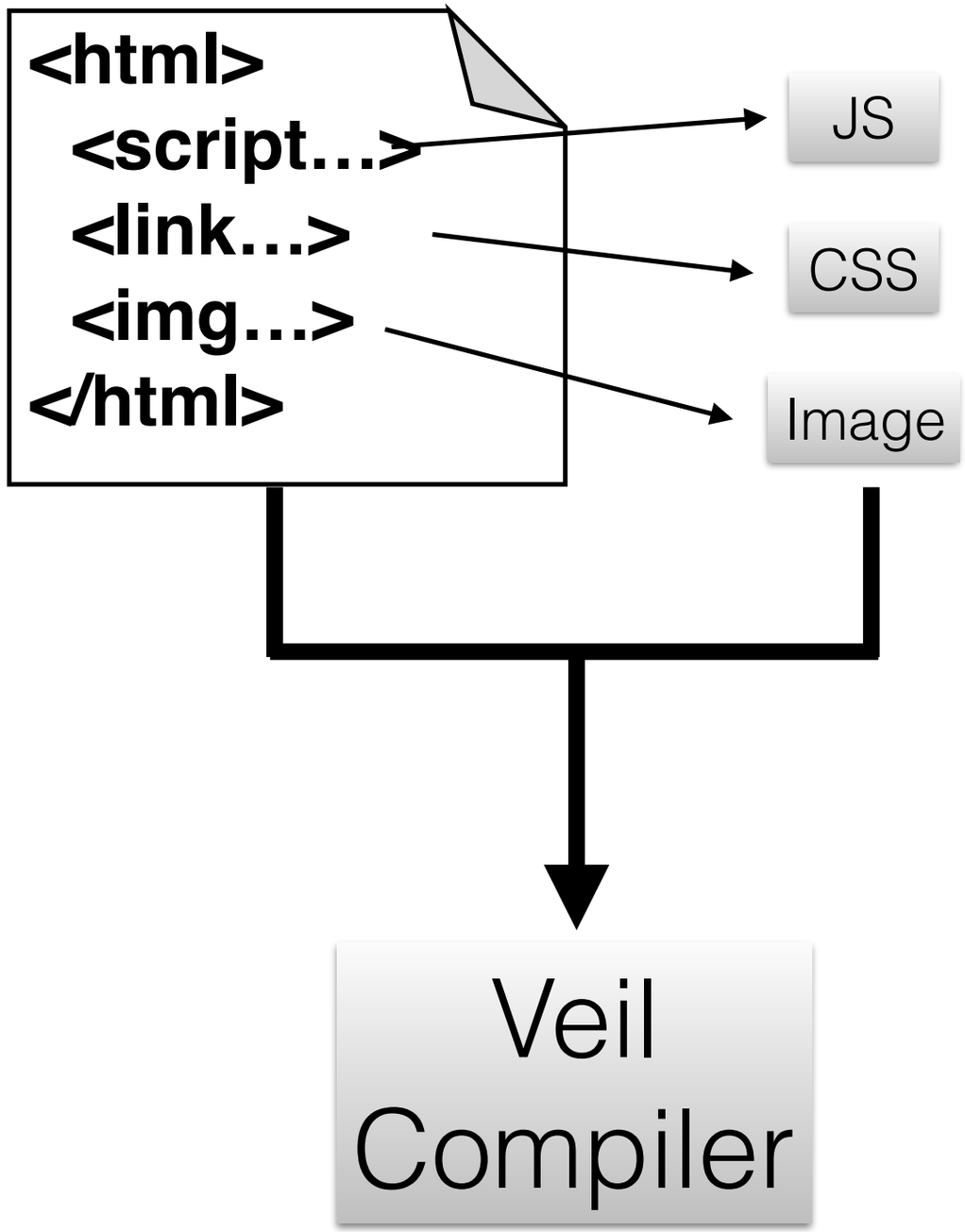
URL blinding,
Content mutation

Client browser

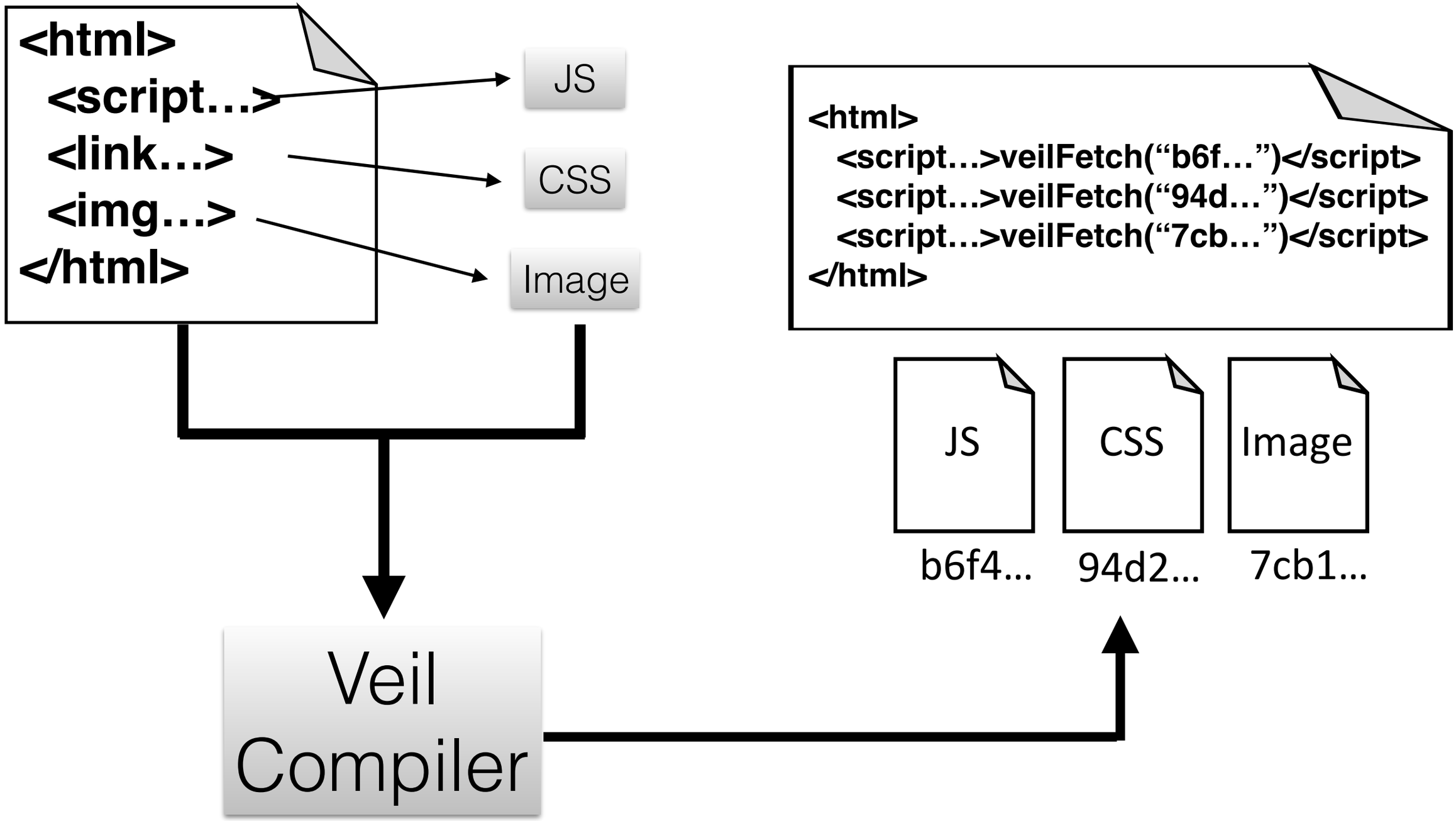
Veil Compiler



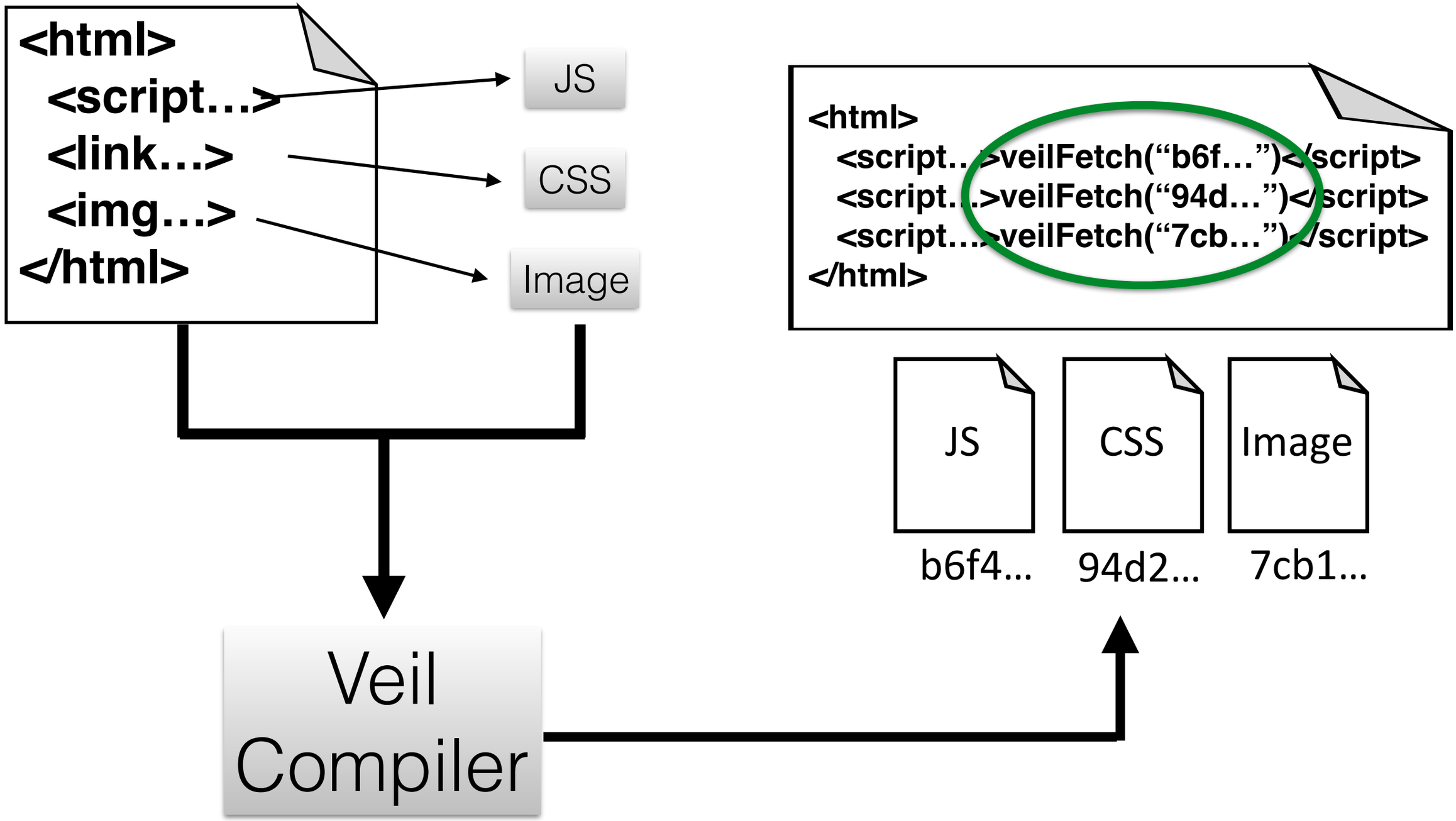
Veil Compiler



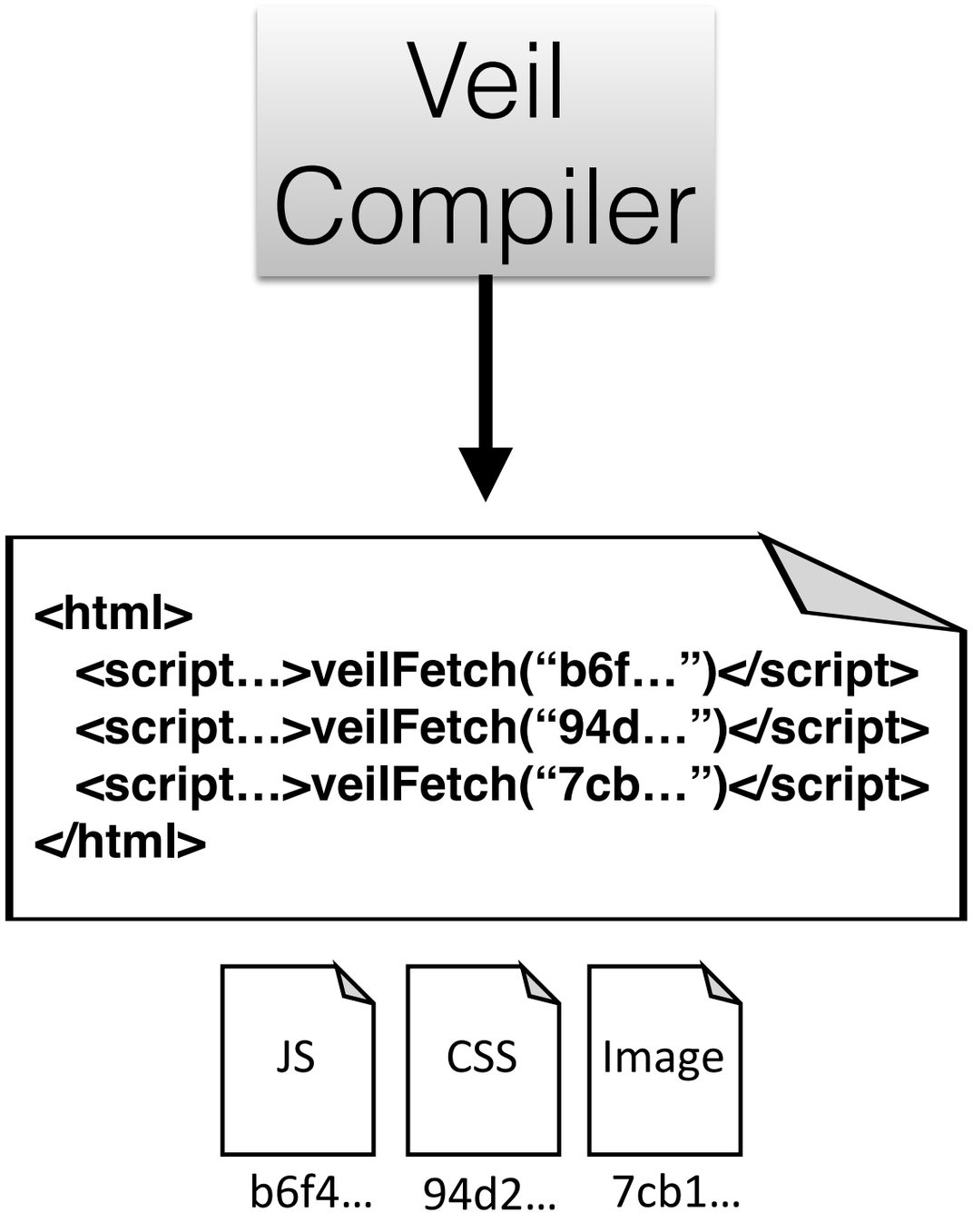
Veil Compiler



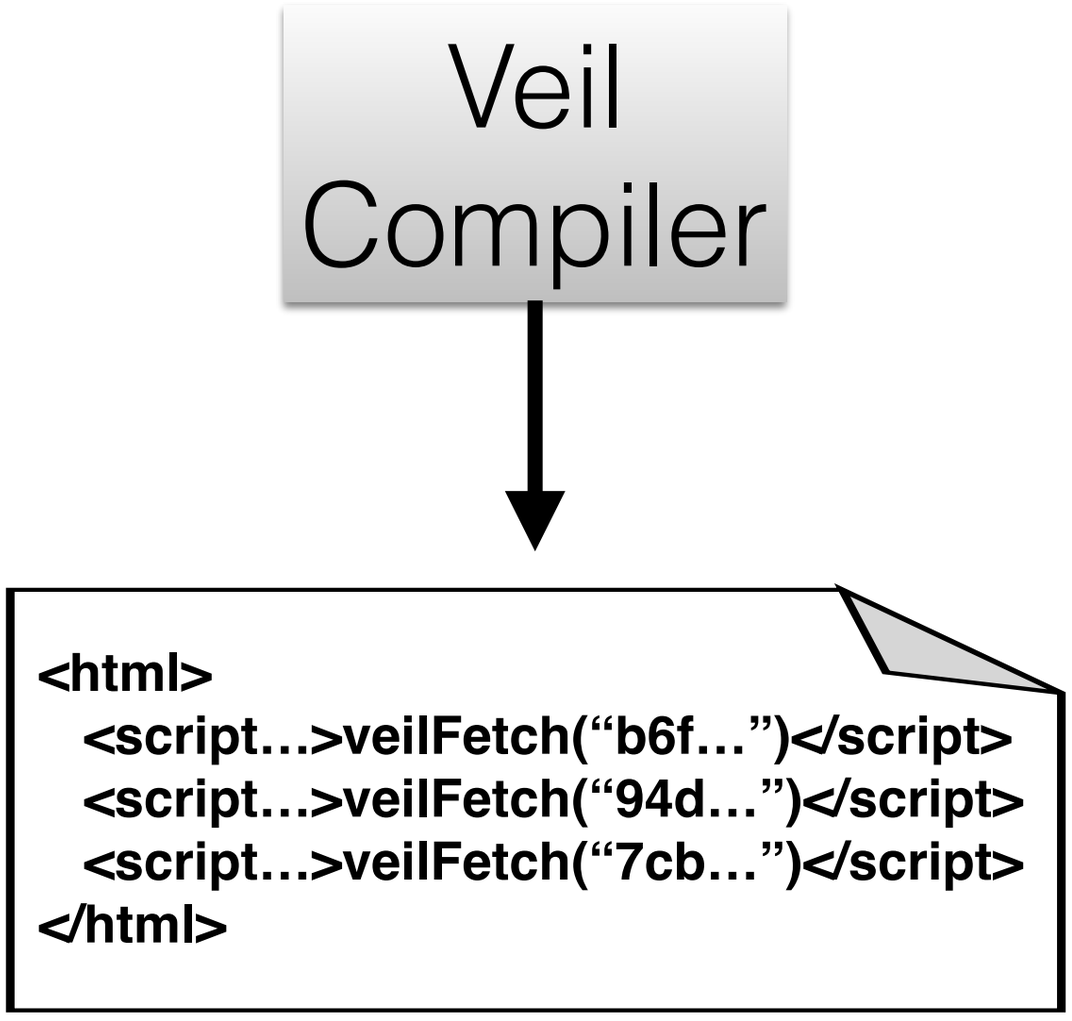
Veil Compiler



Blinding Servers



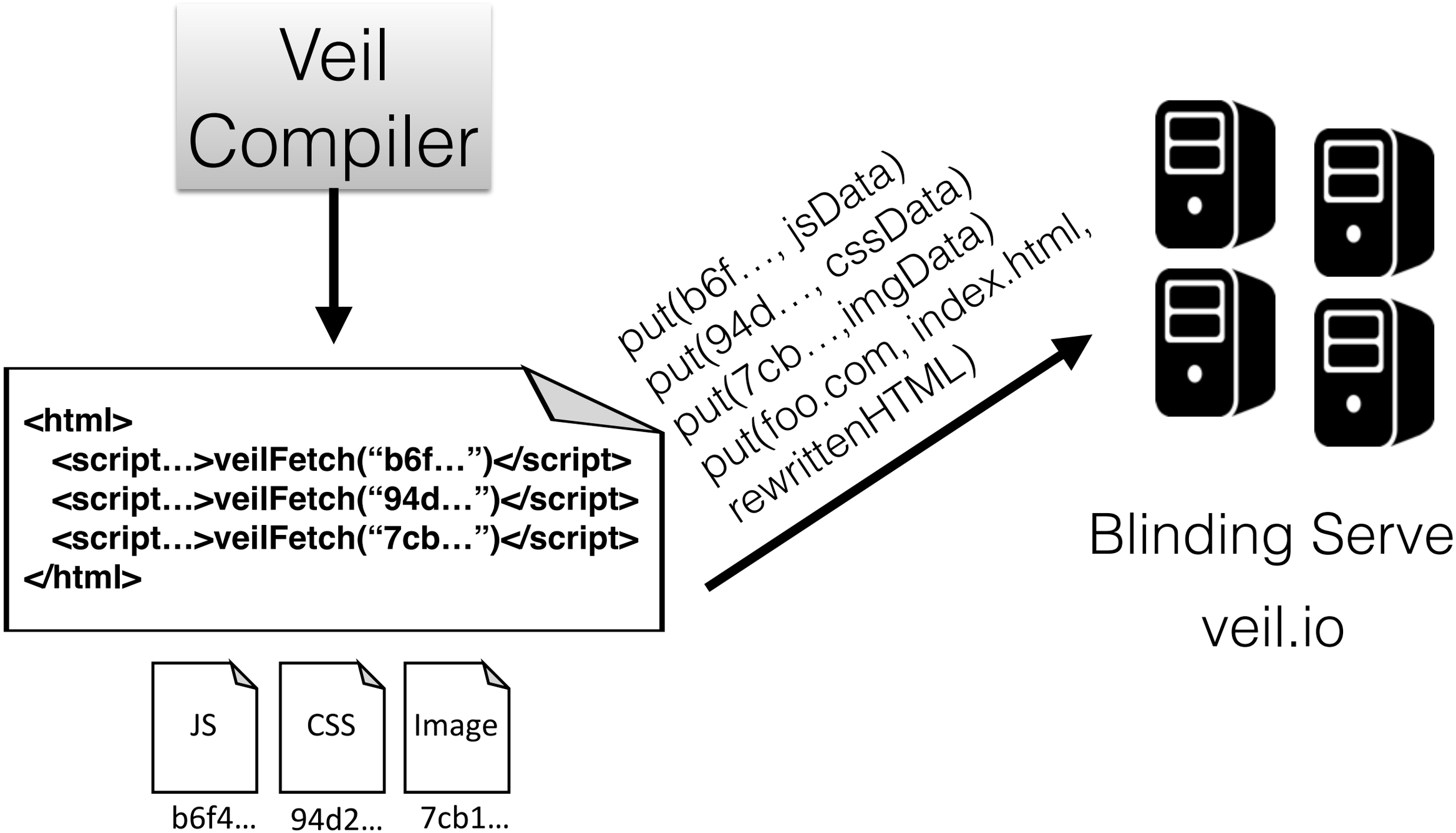
Blinding Servers



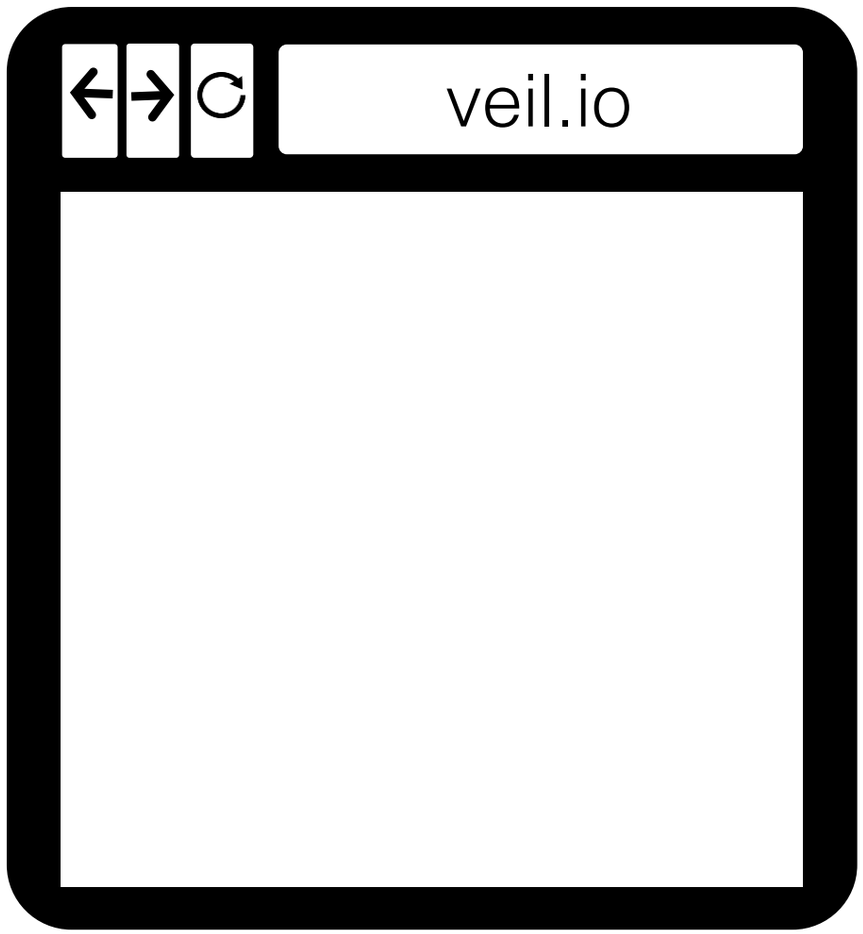
Blinding Servers

veil.io

Blinding Servers

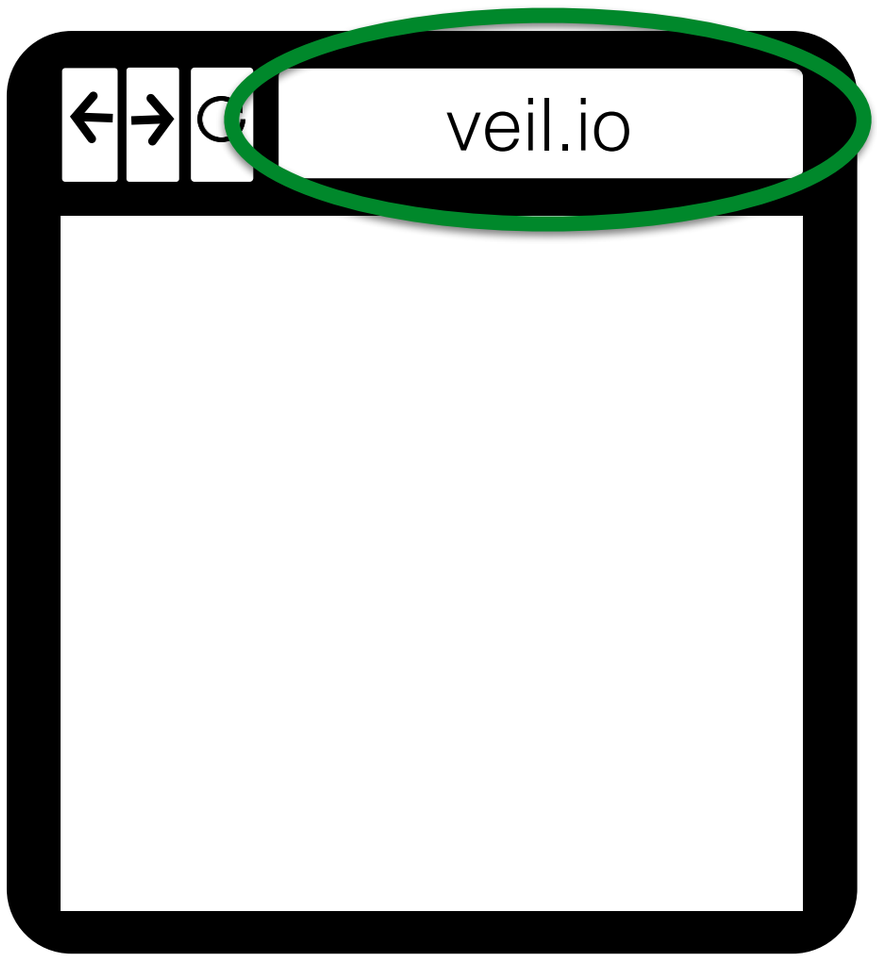


Client Browser: Dynamic Reassembly



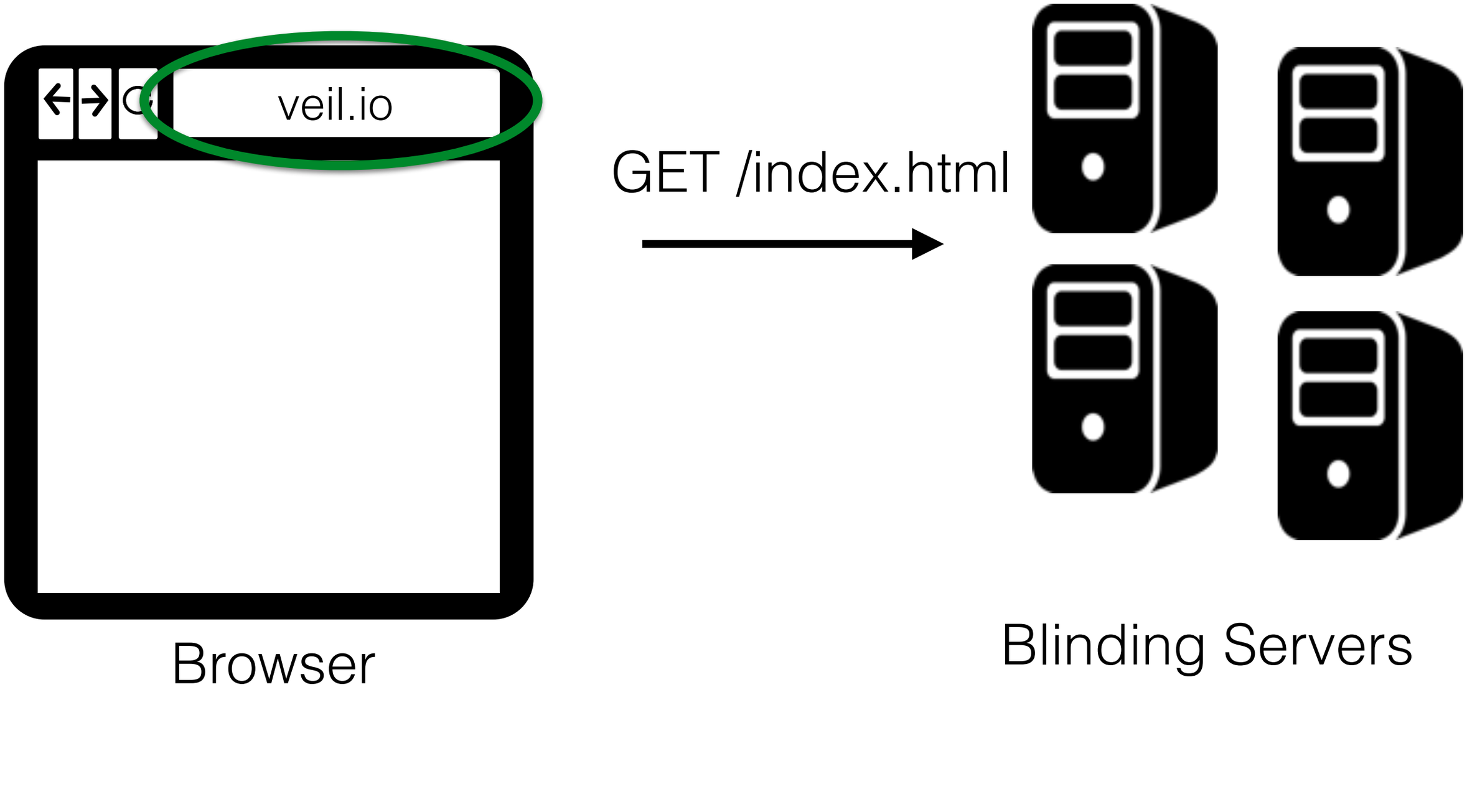
Browser

Client Browser: Dynamic Reassembly

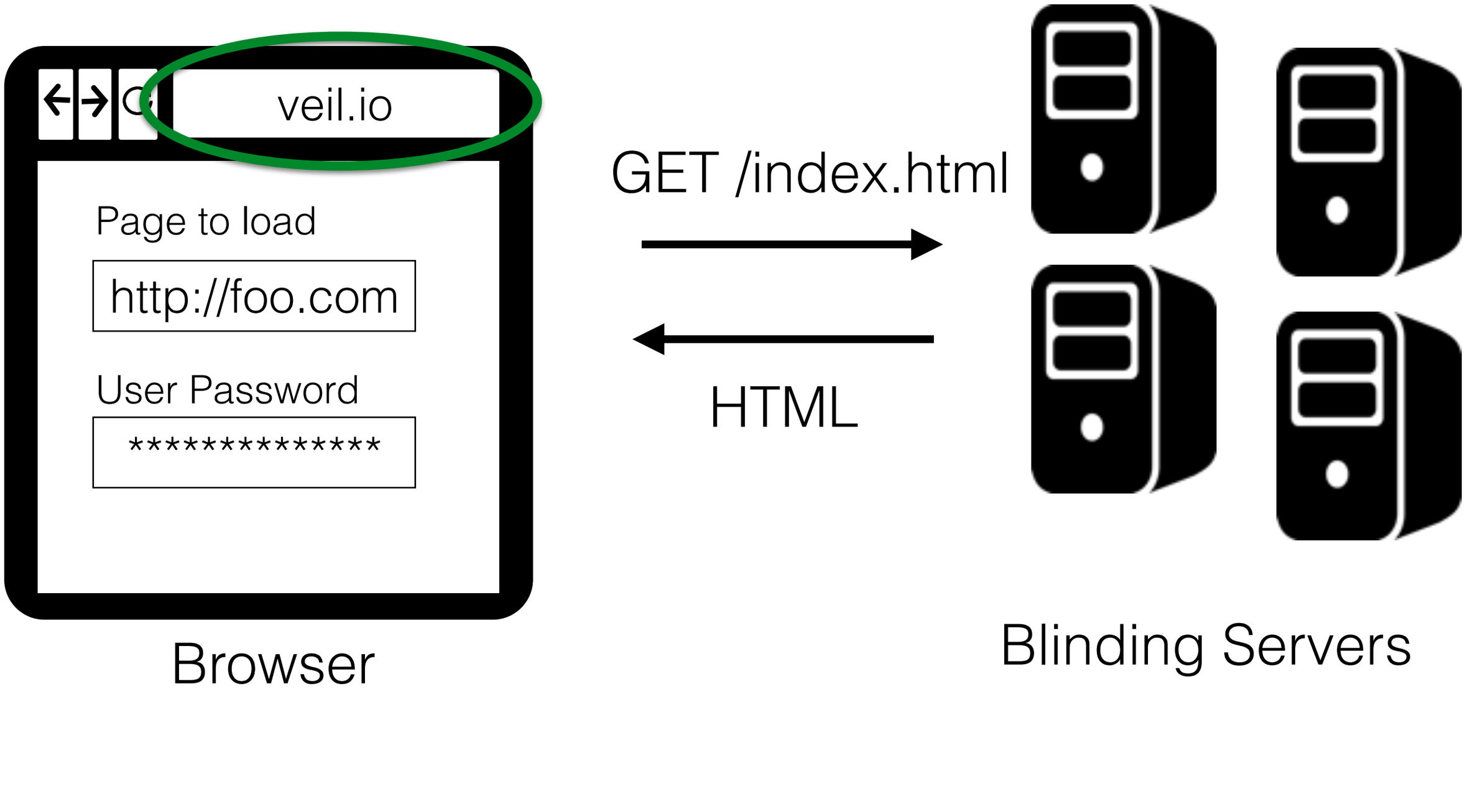


Browser

Client Browser: Dynamic Reassembly



Client Browser: Dynamic Reassembly



Client Browser: Dynamic Reassembly

Browser

Client Browser: Dynamic Reassembly

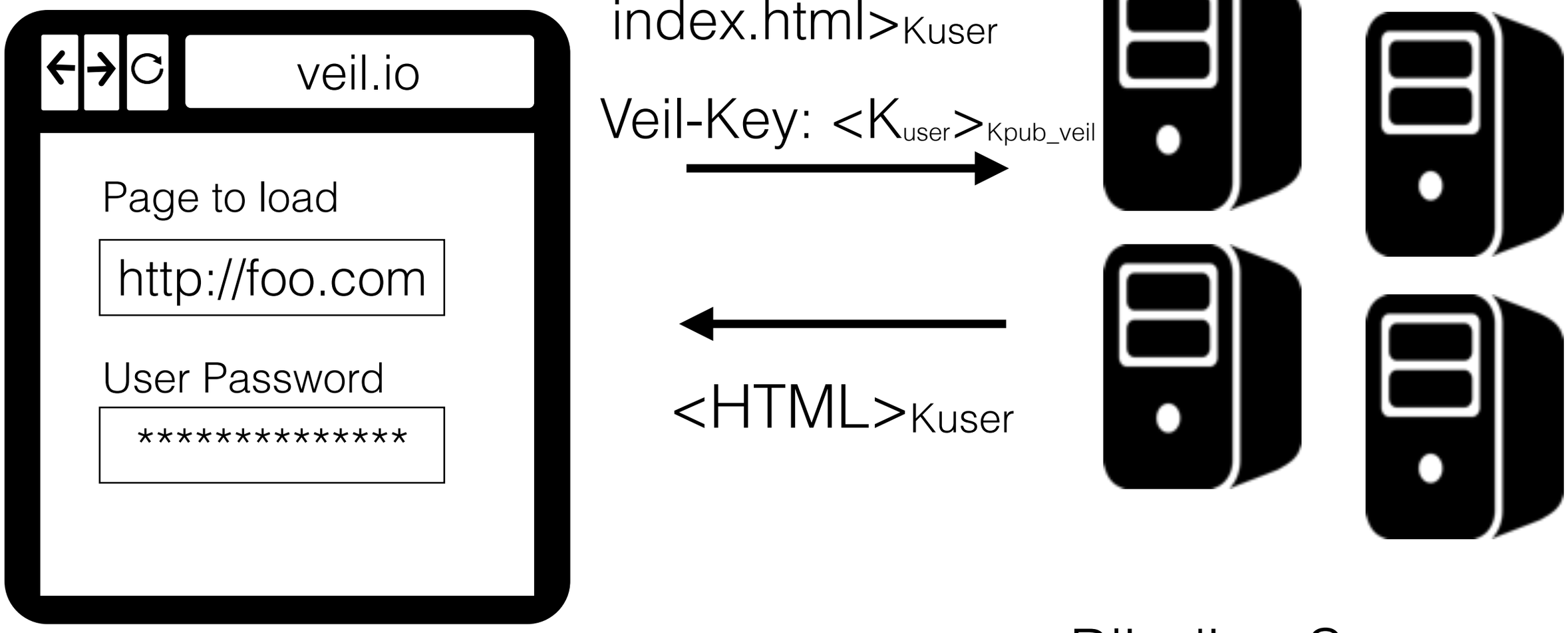
Browser

GET <foo.com/index.html>$_{Kuser}$

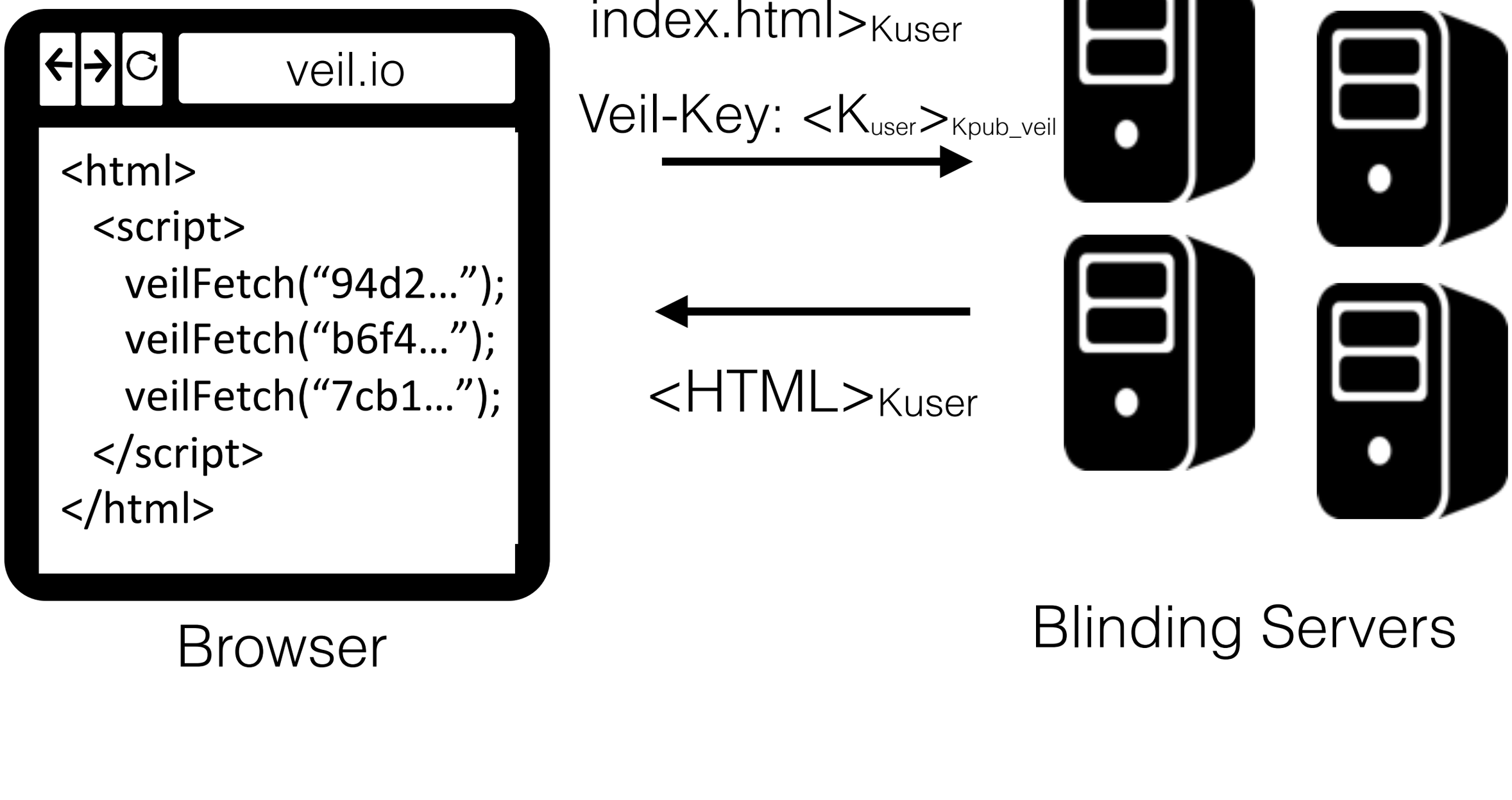
Veil-Key: <K_{user}>$_{Kpub\_veil}$

Blinding Servers

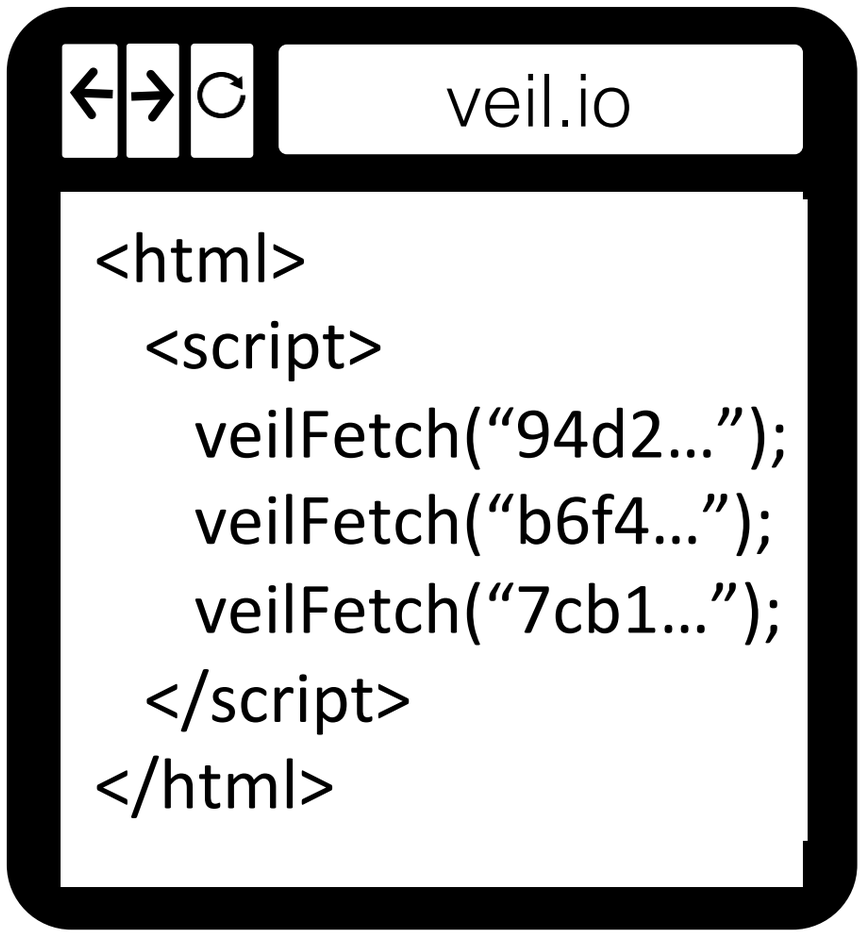
Client Browser: Dynamic Reassembly



Client Browser: Dynamic Reassembly



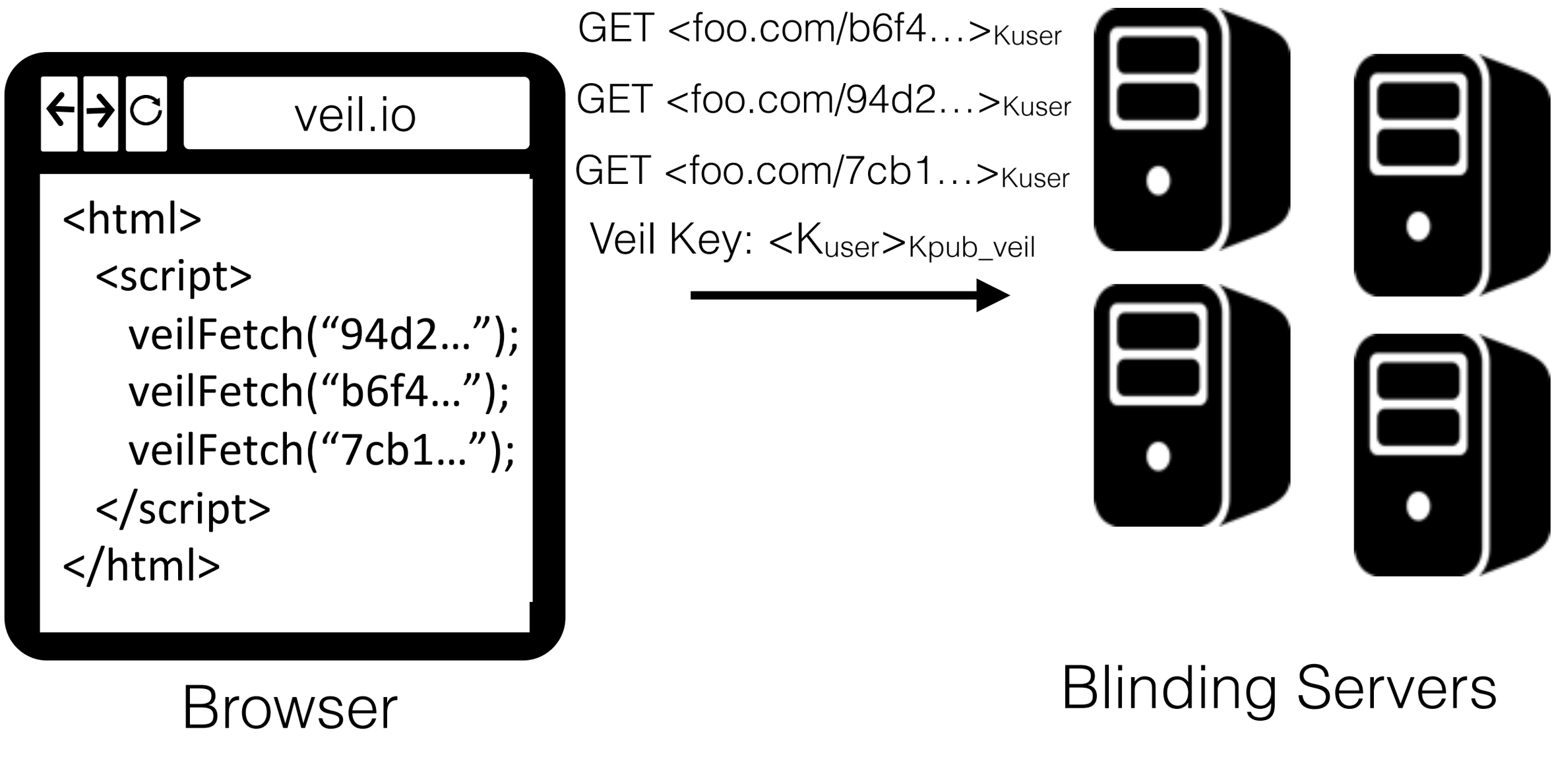
Client Browser: Dynamic Reassembly



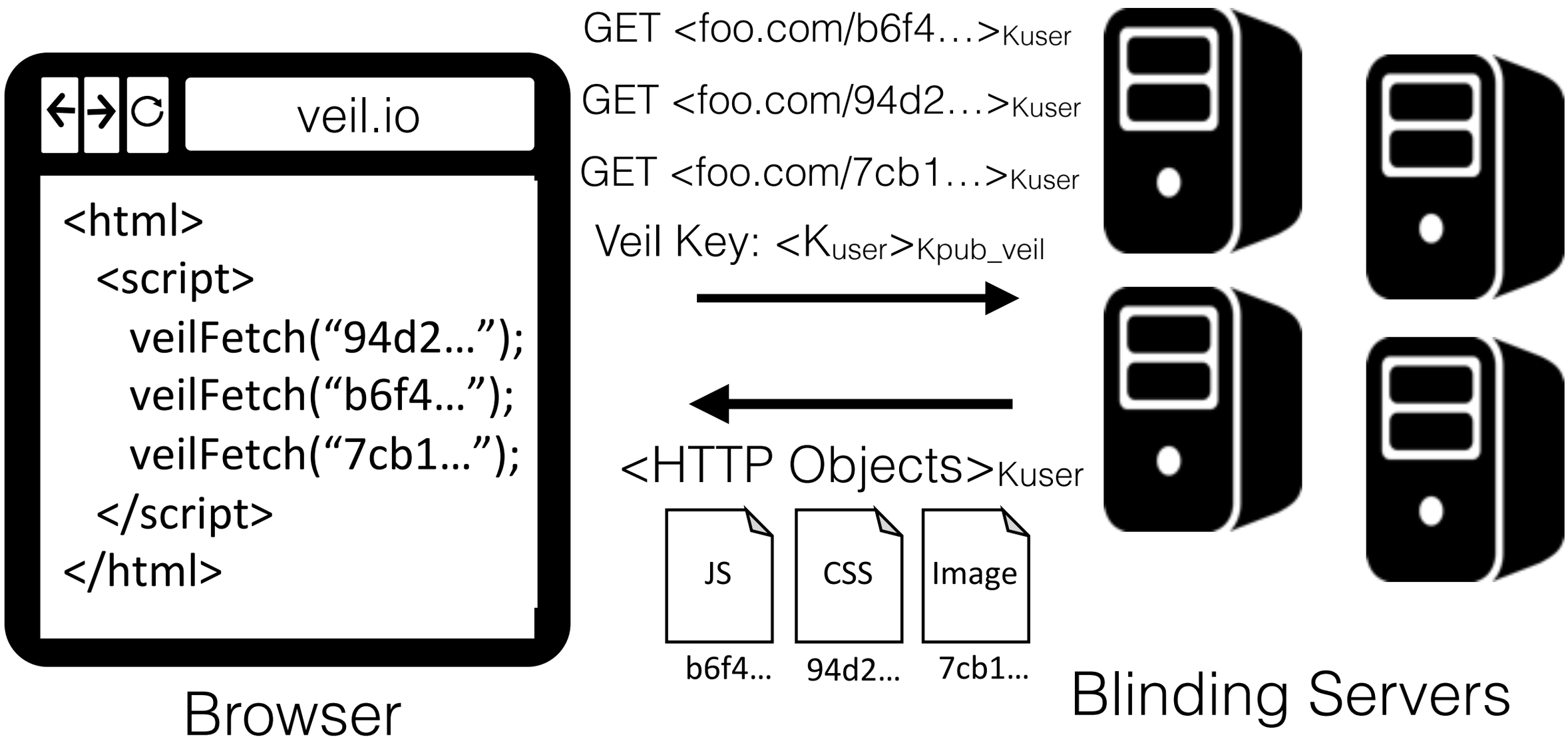
Browser

Blinding Servers

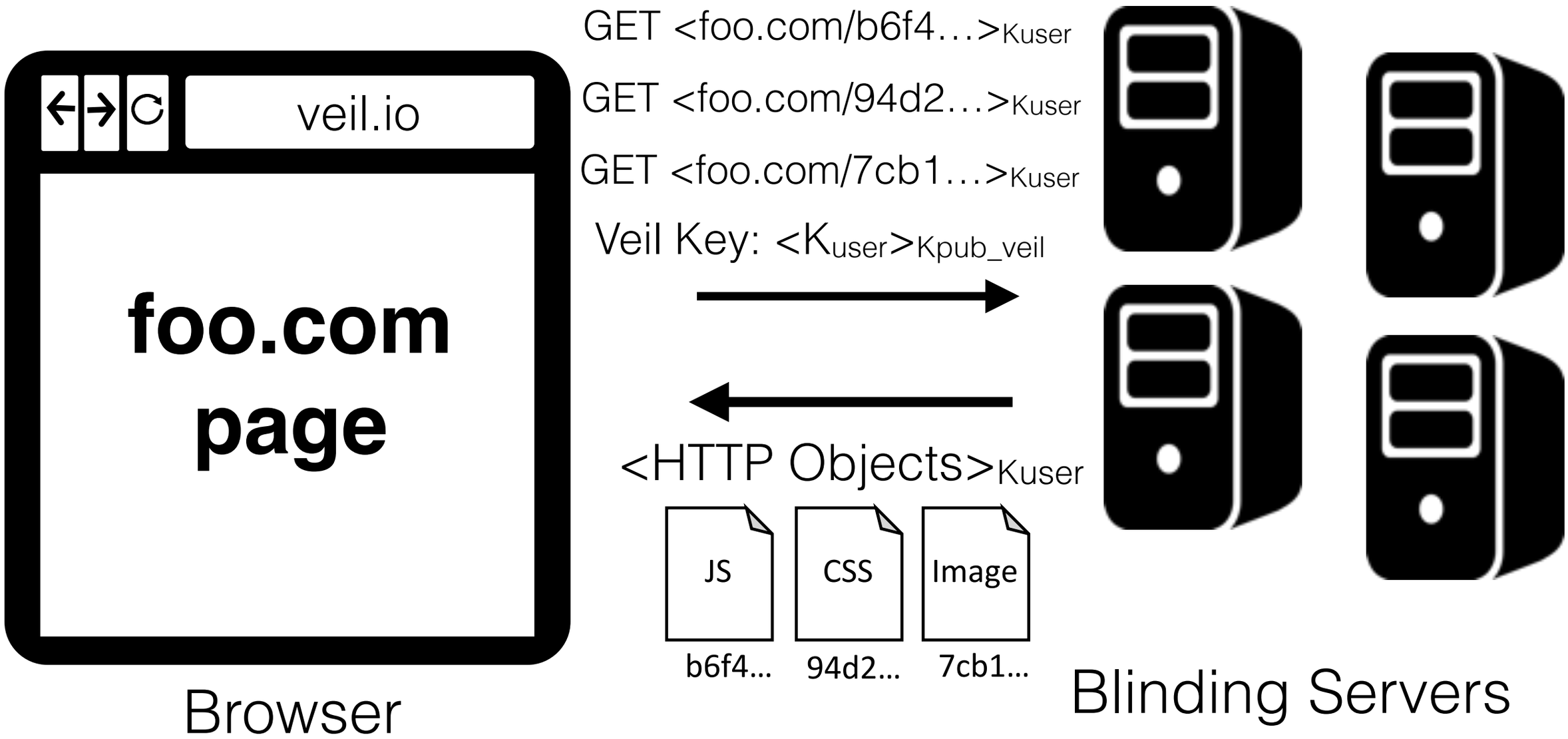
Client Browser: Dynamic Reassembly



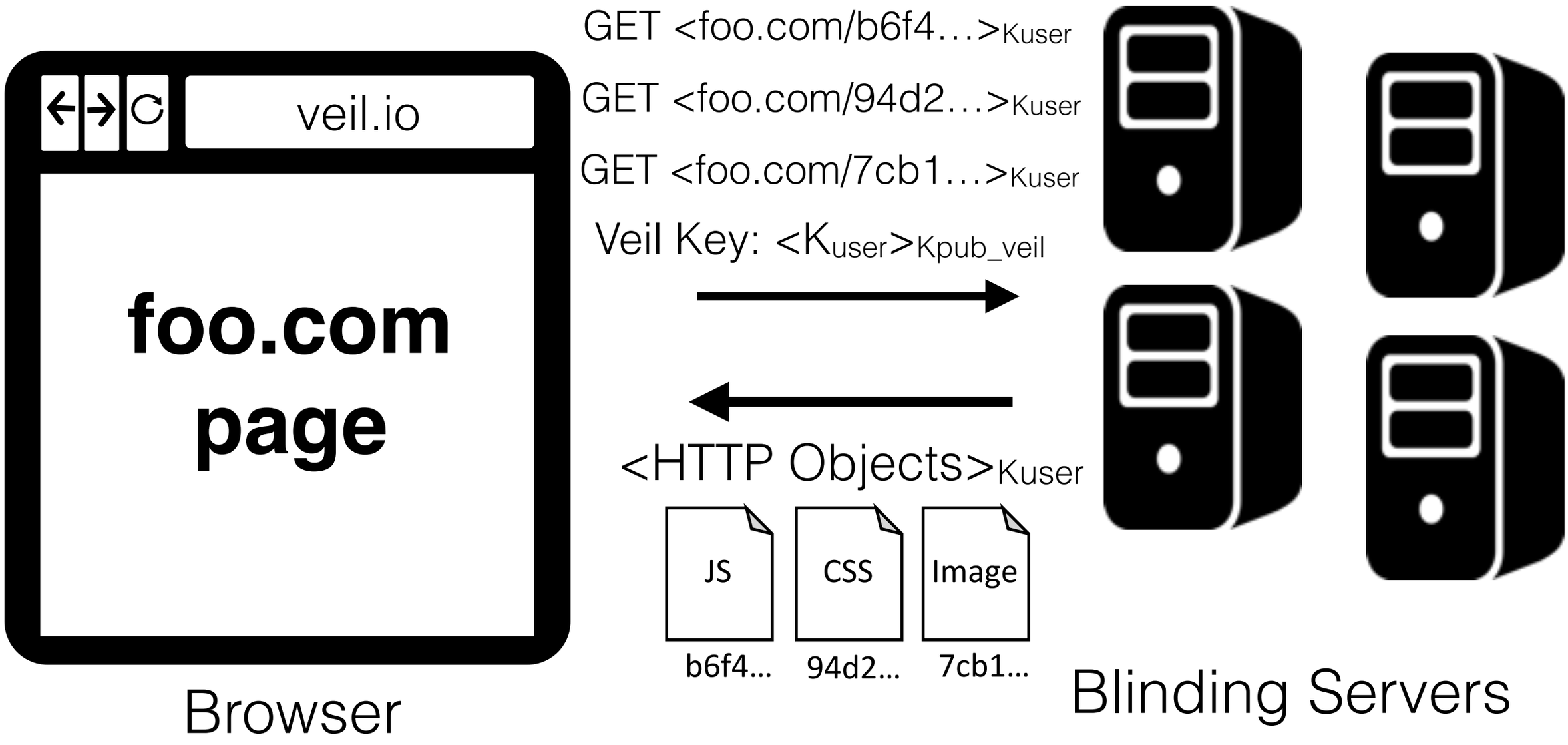
Client Browser: Dynamic Reassembly



Client Browser: Dynamic Reassembly



Client Browser: Dynamic Reassembly



Key Technique: Blinded URLs

Protecting RAM Artifacts

- Heap Walking: reduce likelihood of swap rooted at `markAsSensitive()` tree
- Content Mutation: not leak site-specific content
  - Noise to images
  - Add junk code
  - `eval()`-folding
- More details in the paper

Protecting RAM Artifacts

- Heap Walking: reduce likelihood of swap rooted at `markAsSensitive()` tree
- Content Mutation: not leak site-specific content
  - Noise to images
  - Add junk code
  - `eval()`-folding
- More details in the paper

URL blinding, heap walking, and content mutation are inherently unimplementable by the browser!

Problem: Complex DOM structure still exists

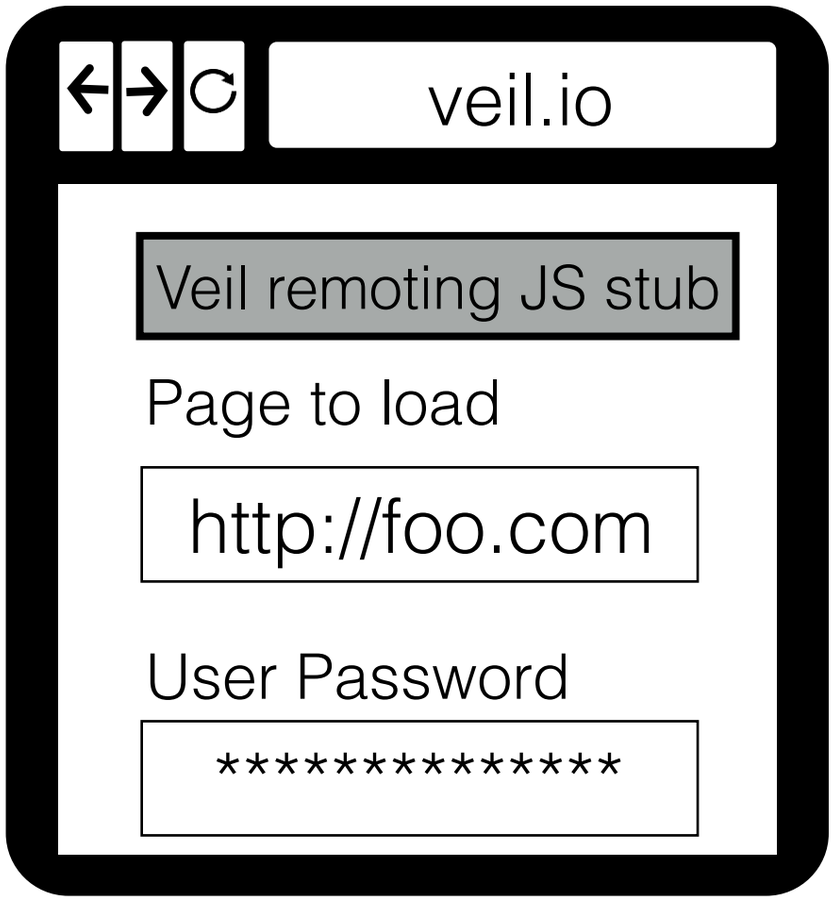
Problem: Complex DOM structure still exists

- Heap walking reduces likelihood of swapping
- Content mutation makes it difficult to recover
- However, some sites might want to minimize client-side DOM state

Problem: Complex DOM structure still exists

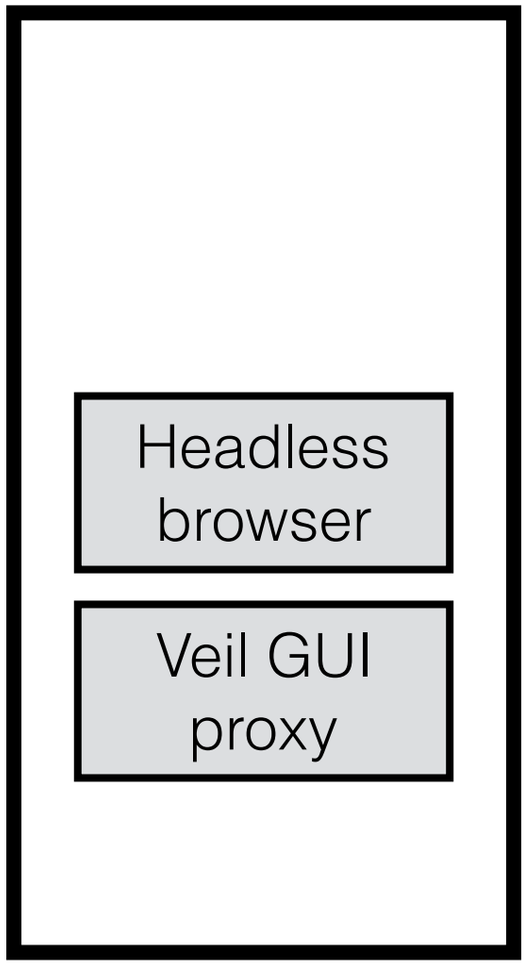
- Heap walking reduces likelihood of swapping
- Content mutation makes it difficult to recover
- However, some sites might want to minimize client-side DOM state
- Solution: DOM hiding mode
  - User's browser as a thin client
  - Remote server loads real page, applies GUI event, and returns screenshot of updated page

DOM Hiding Mode



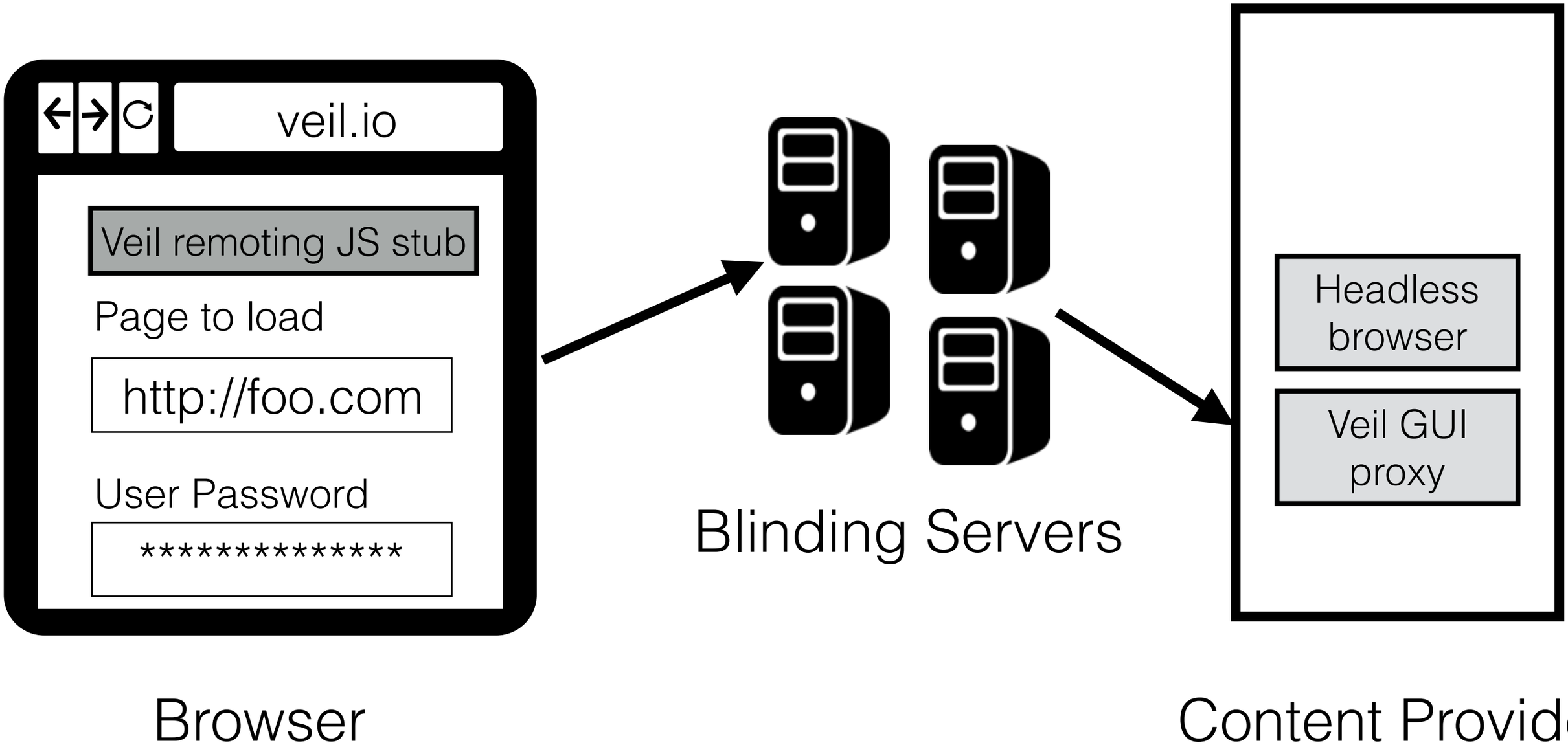
Browser

Blinding Servers

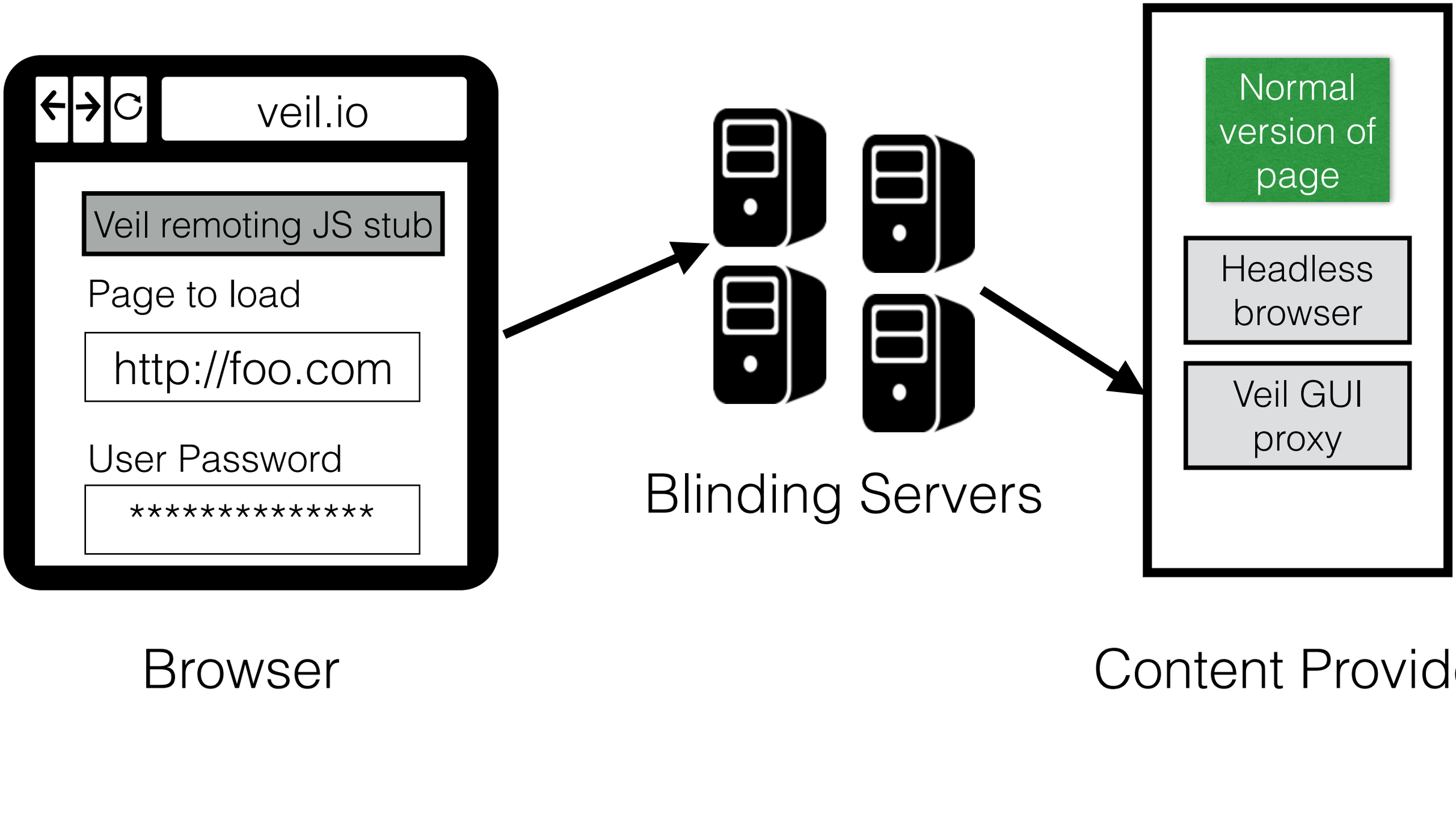


Content Provider

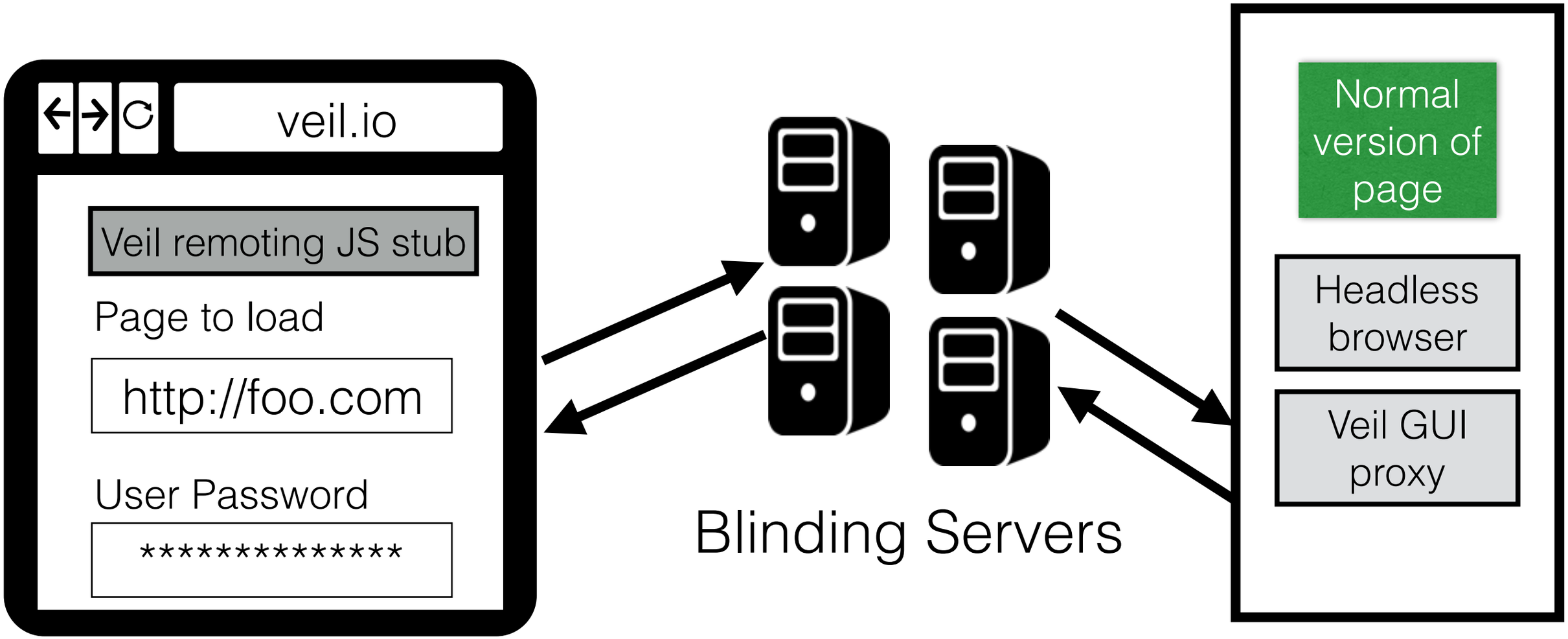
DOM Hiding Mode



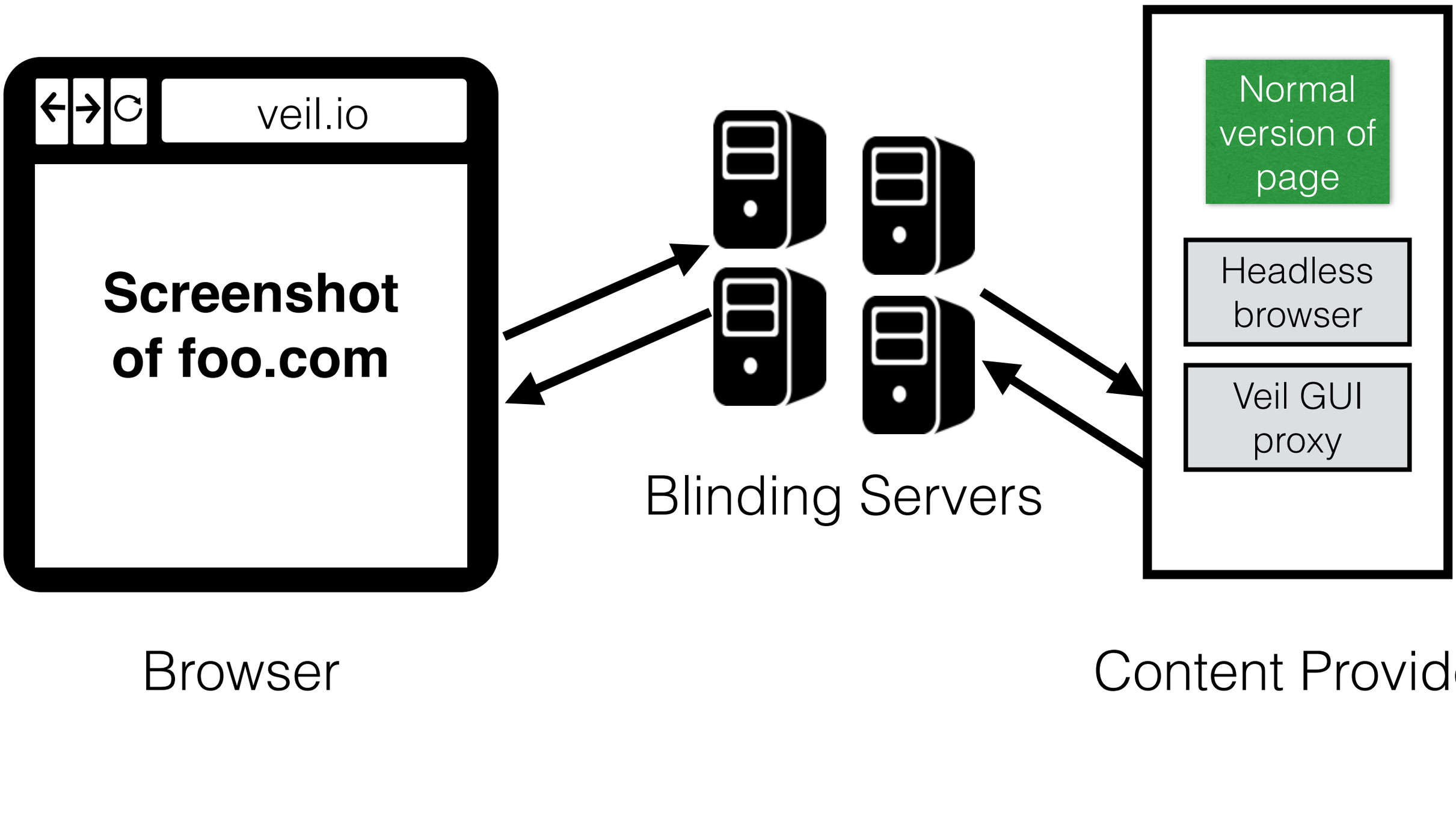
DOM Hiding Mode



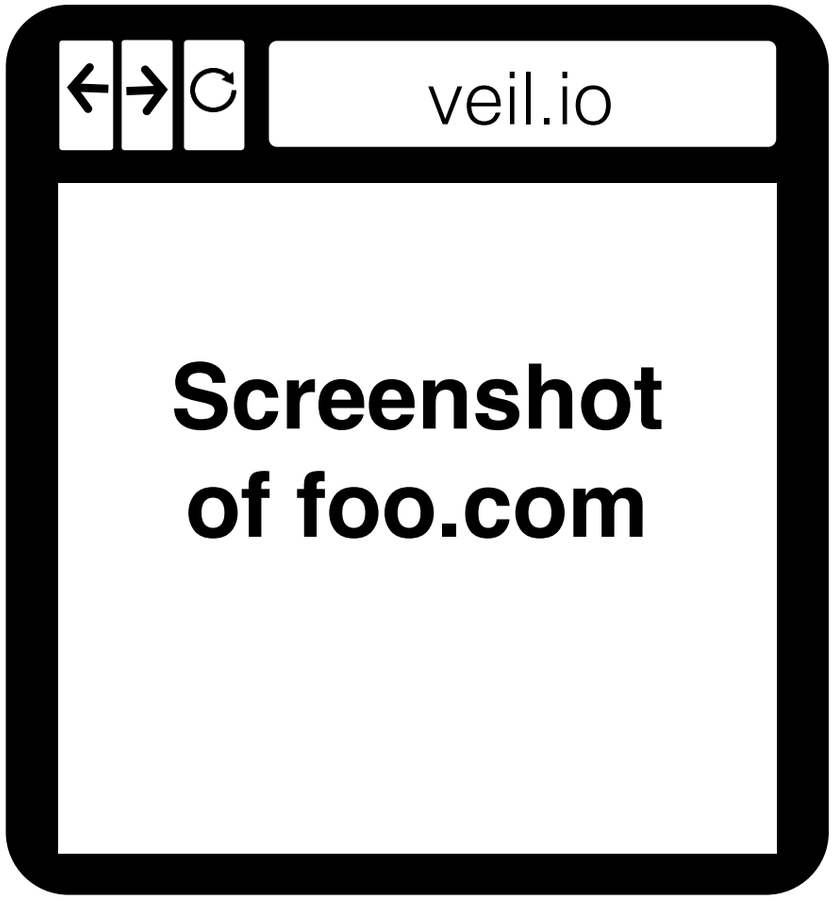
DOM Hiding Mode



DOM Hiding Mode

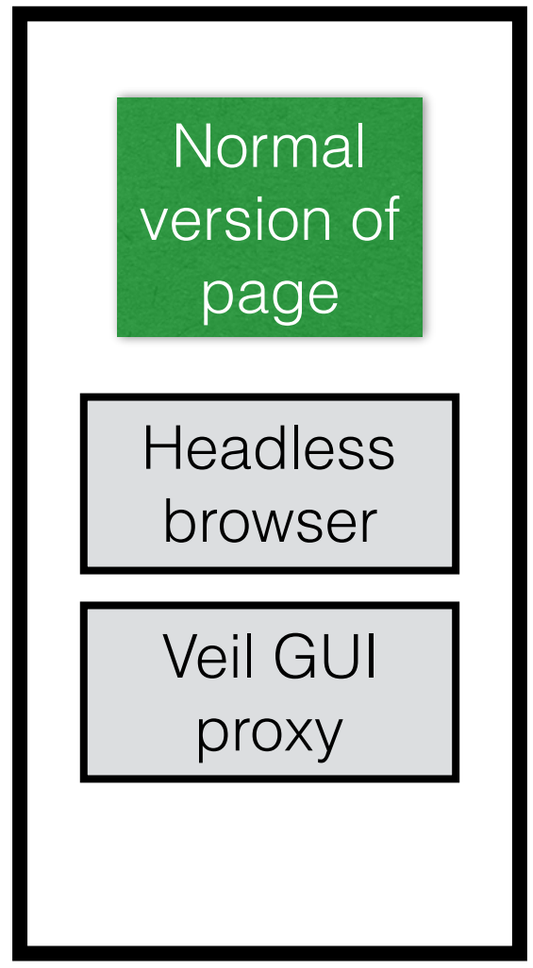


DOM Hiding Mode



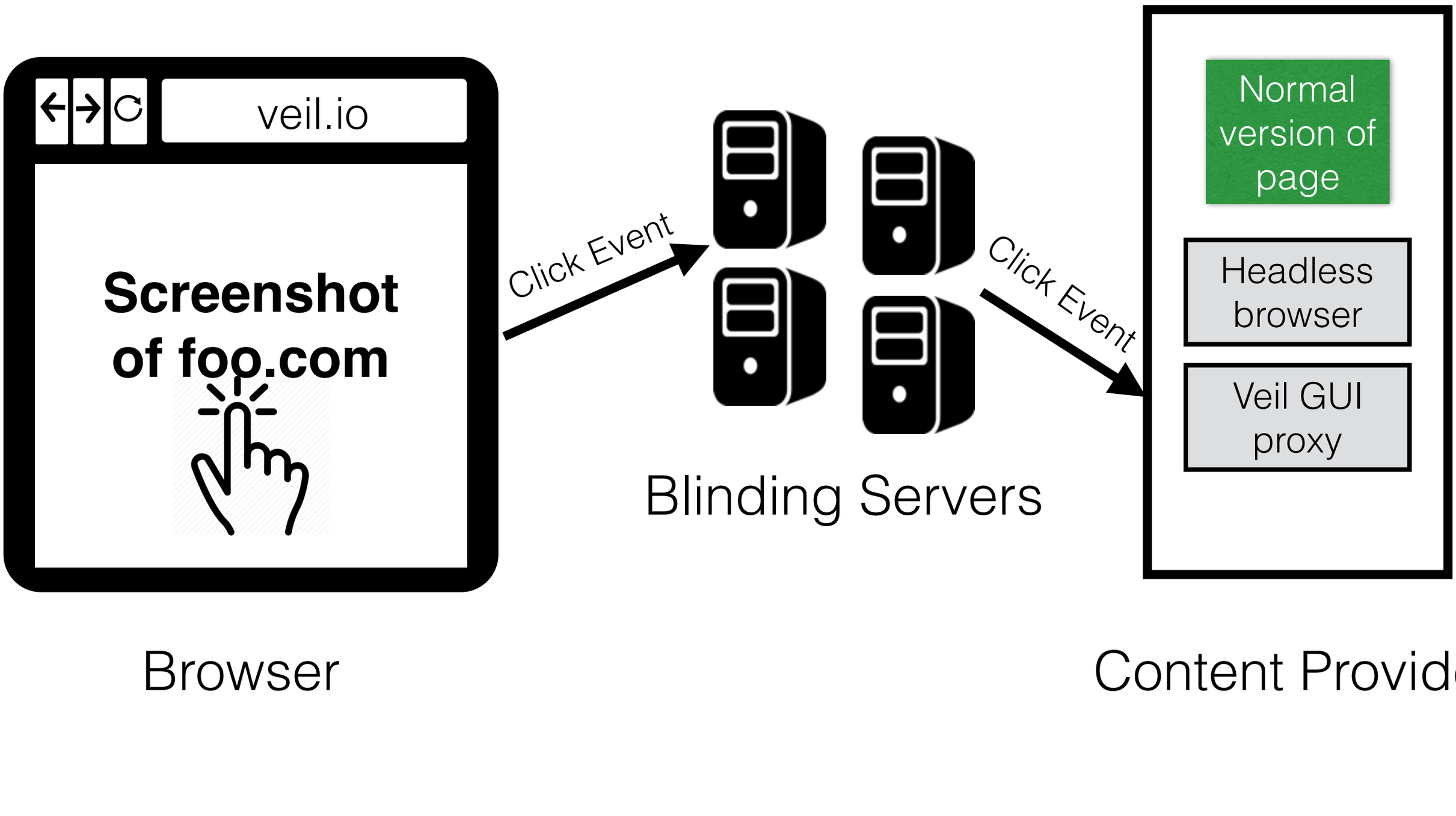
Browser

Blinding Servers

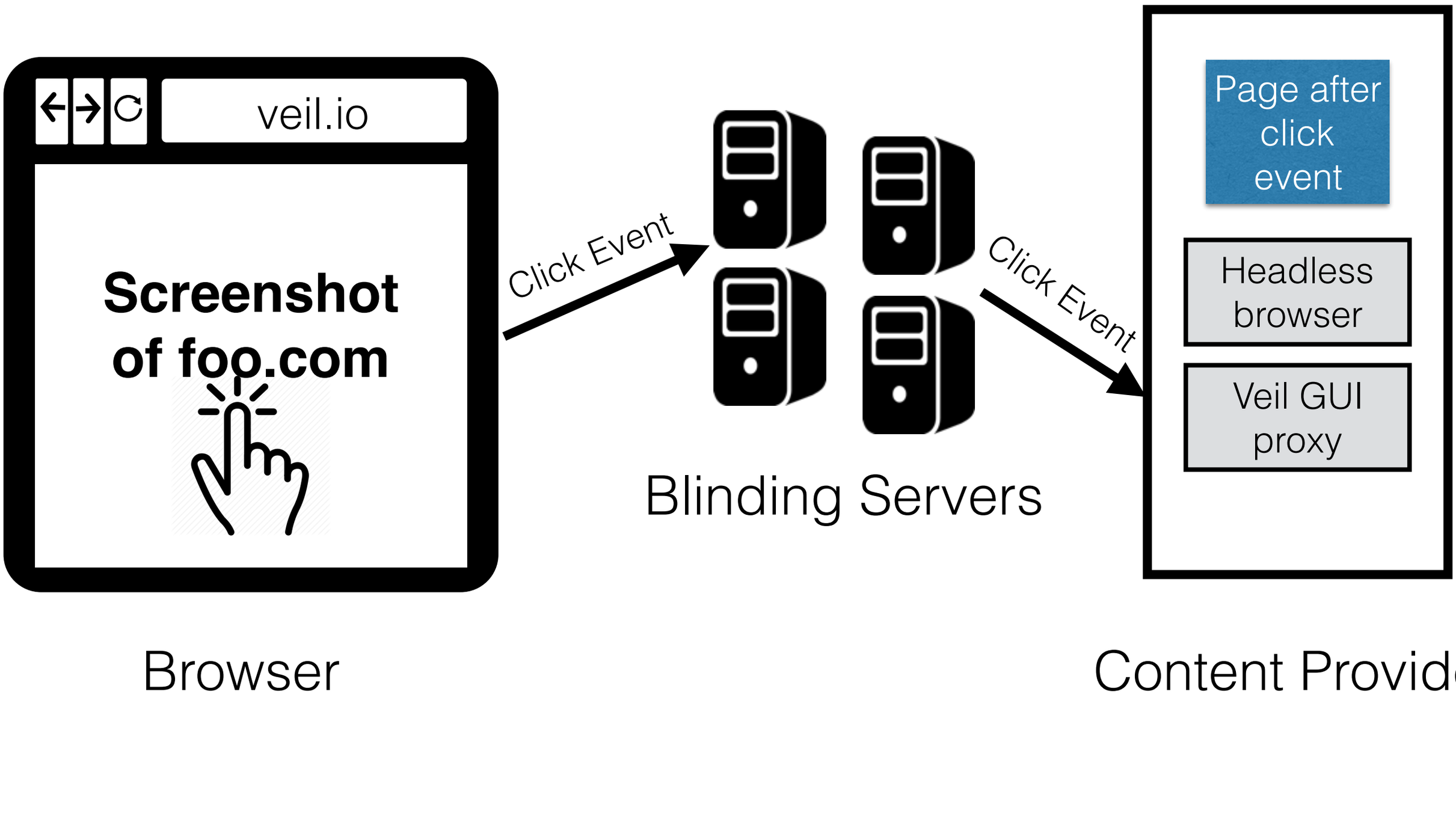


Content Provider

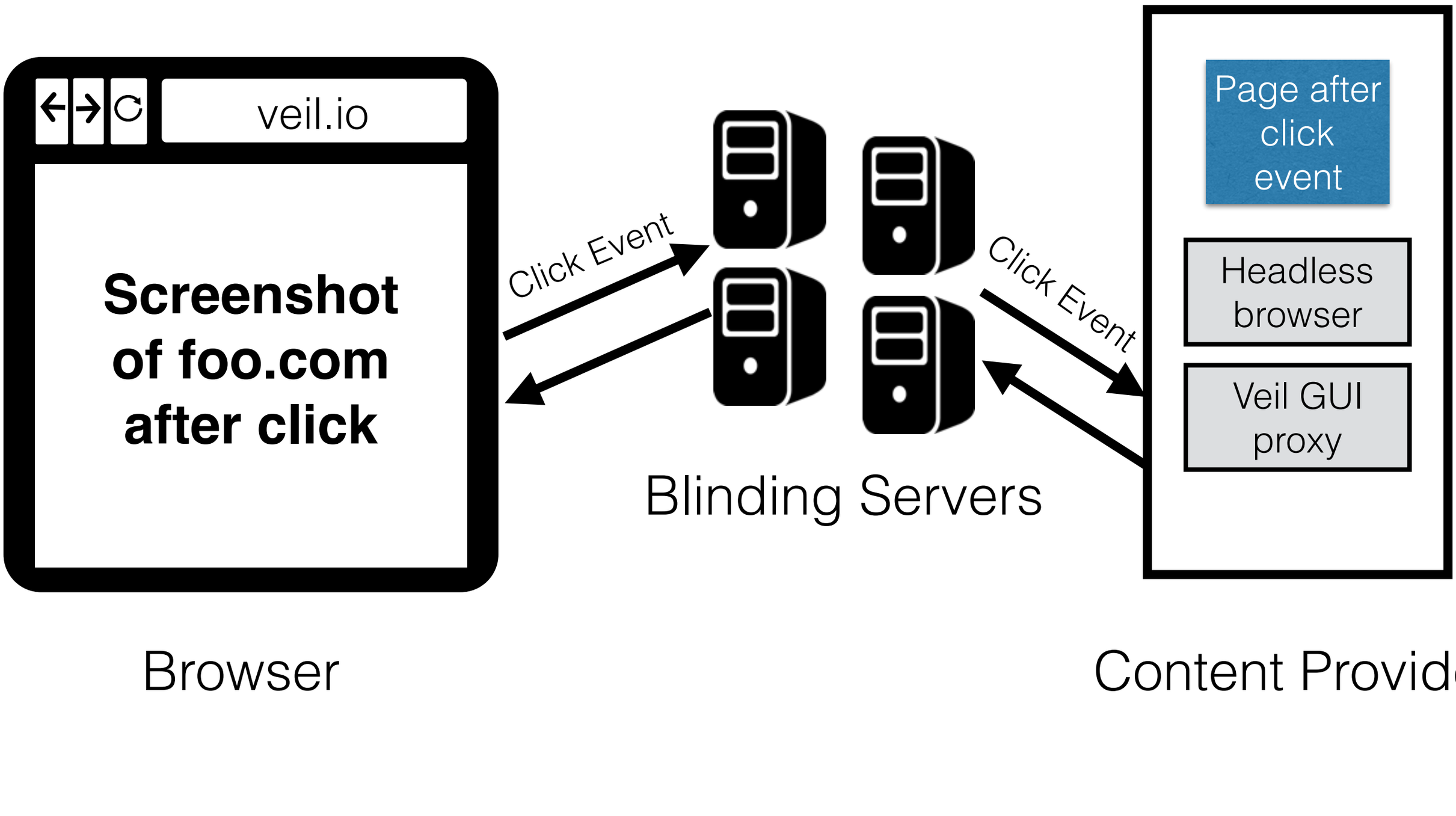
DOM Hiding Mode



DOM Hiding Mode



DOM Hiding Mode



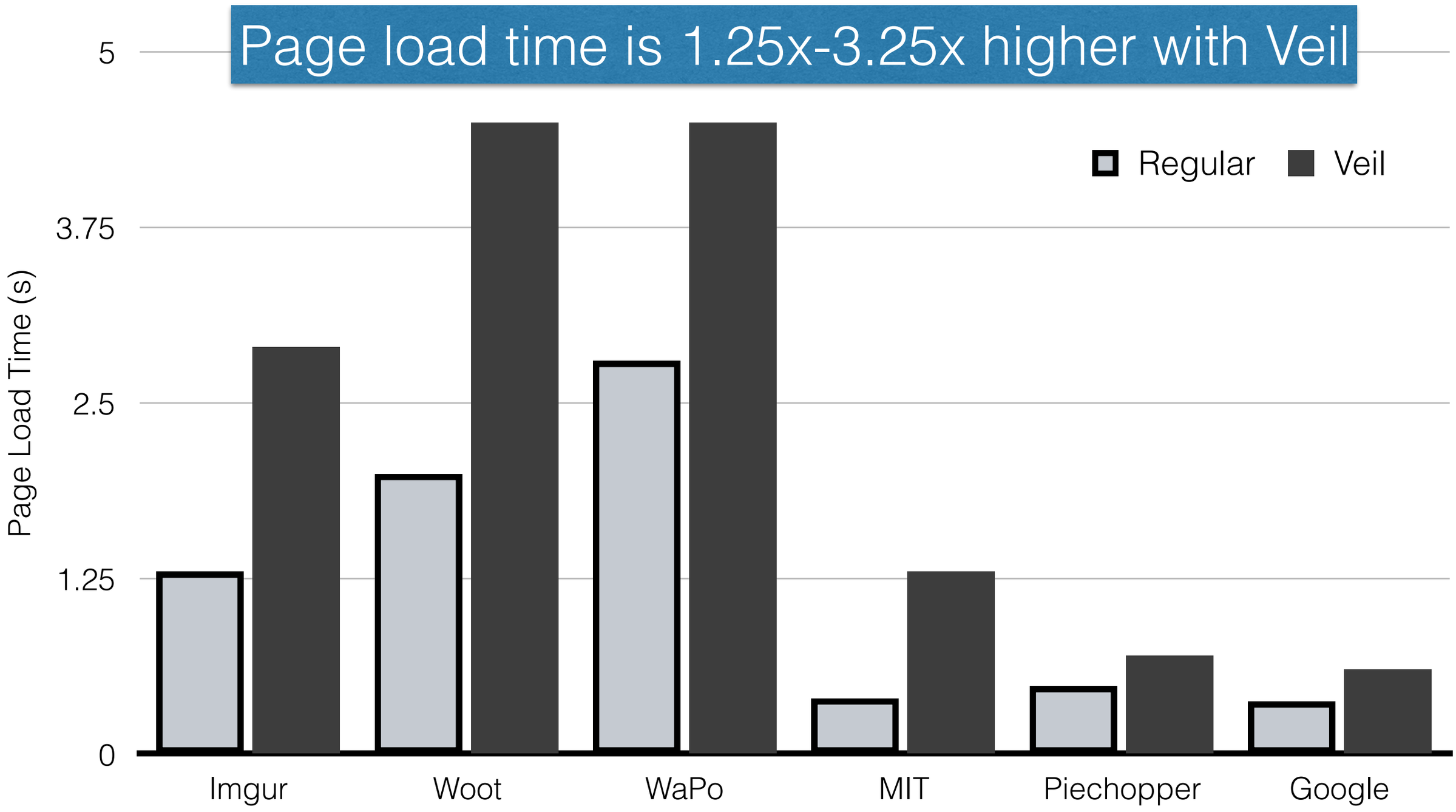
Outline

- Veil Architecture
- **Implementation**
- **Evaluation**

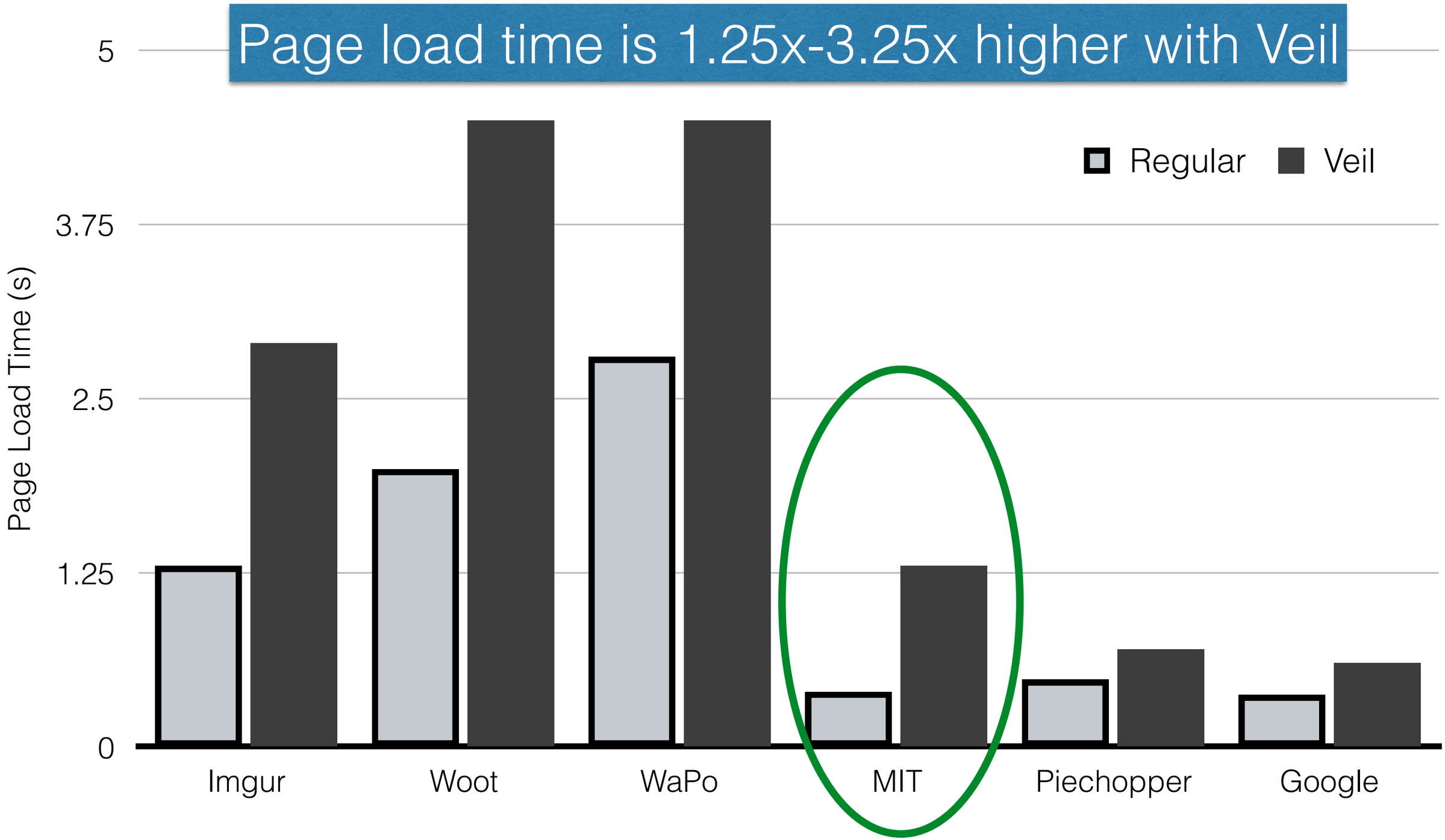
Implementation

- 4 components
  - Compiler
  - Blinding servers
  - JS library for bootstrap page and DOM hiding mode
  - GUI proxy (for DOM hiding mode)
- Compiler and blinding server written in Python
- GUI proxy uses headless Chrome
- BeautifulSoup to parse and mutate HTML

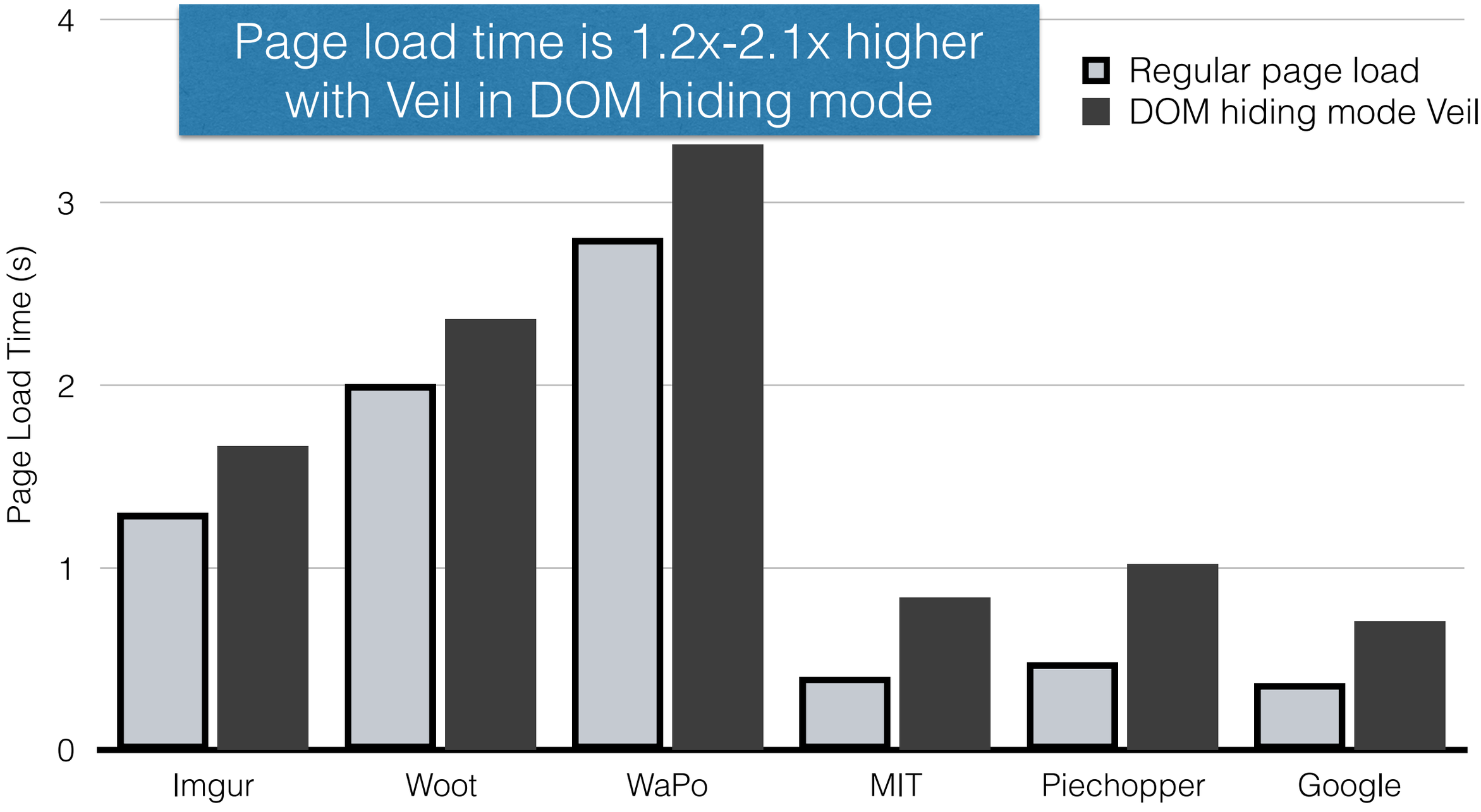
Page Load Times: Standard Veil Mode



Page Load Times: Standard Veil Mode



Page Load Times: Veil in DOM Hiding Mode



Related Work

- CleanOS, Lacuna, PrivExec
  - uses secure deletion to implement privacy
  - require configuration and installation of special runtime
    - cannot protect sensitive data unless abstractions spread across the whole system
- UCognito
  - requires modified client-side stack
  - does not prevent information leakage via non-sandboxed parts

Conclusions

- Traditional private browsing modes still leak information!
- Veil allows developers to improve privacy semantics of their pages
- Veil uses a variety of techniques, which are unimplementable by the browser, to hide sensitive information from post-session attackers
- We evaluated Veil on various real websites and found moderate overhead