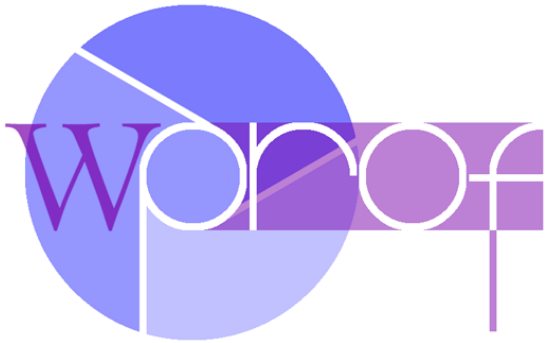
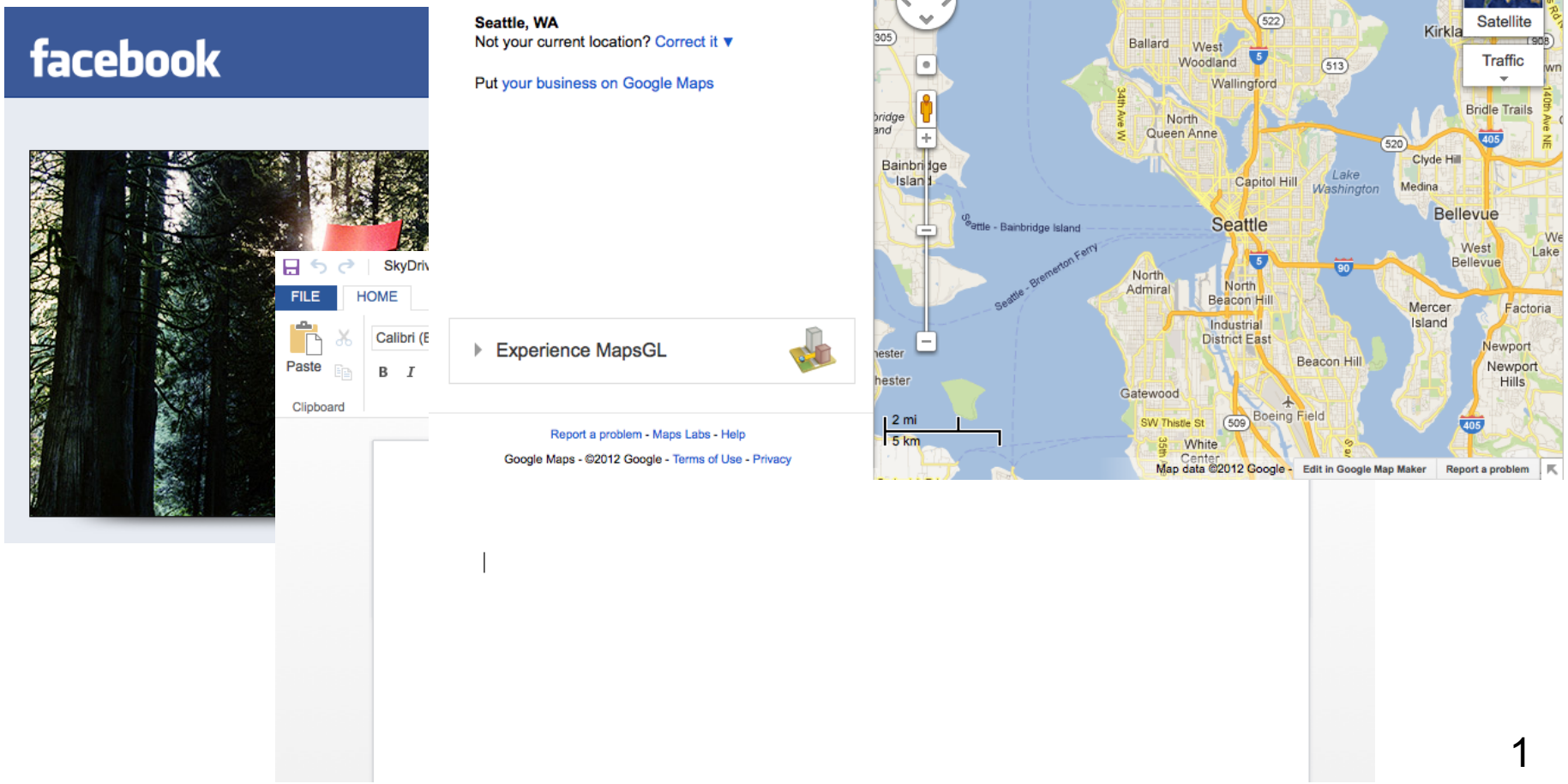
Demystifying Page Load Performance with WProf



Xiao (Sophia) Wang, Aruna Balasubramanian,
Arvind Krishnamurthy, and David Wetherall
University of Washington

Web is the critical part of the Internet

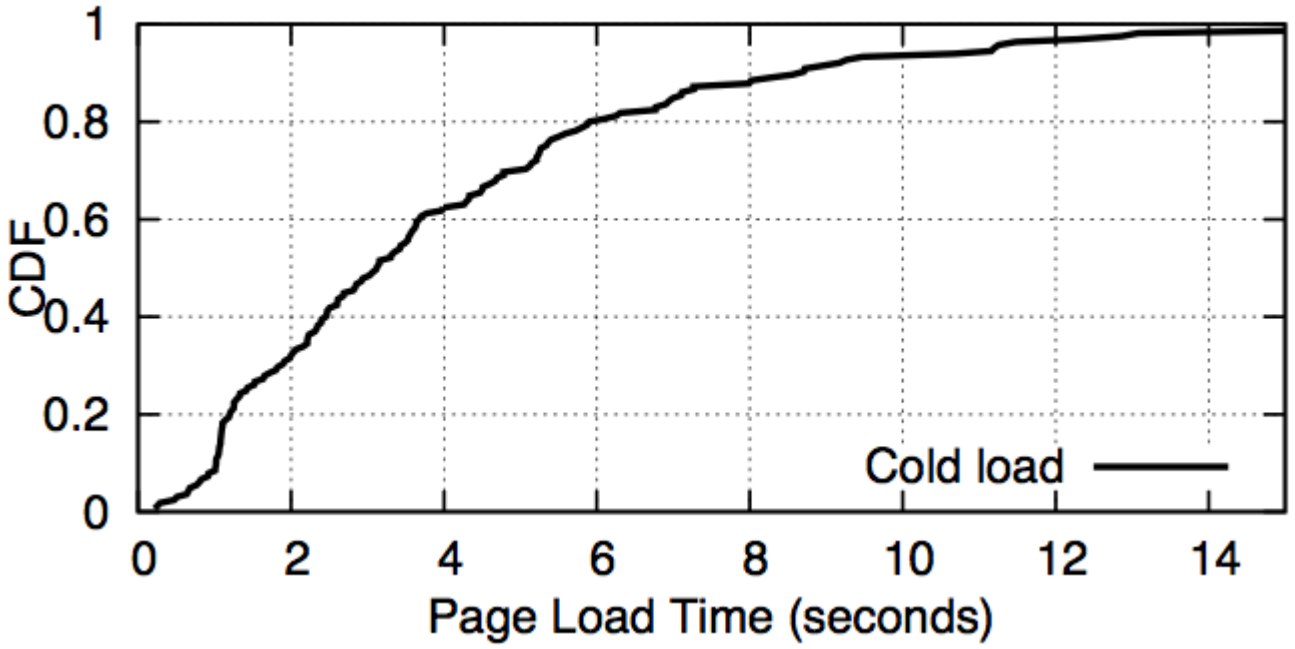


Page load is critical

- Amazon can increase 1% revenue by decreasing page load time by 0.1s.

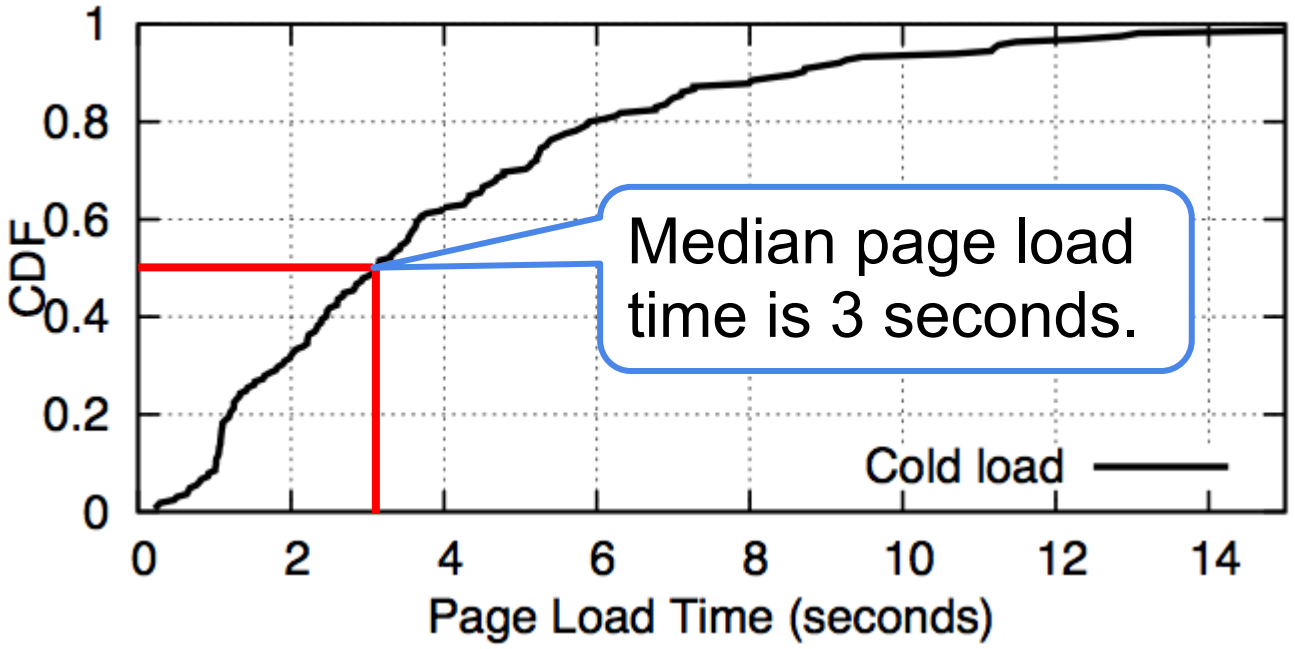
Page load is critical but slow

- Amazon can increase 1% revenue by decreasing page load time by 0.1s.
- Page load is slow even on top 200 websites



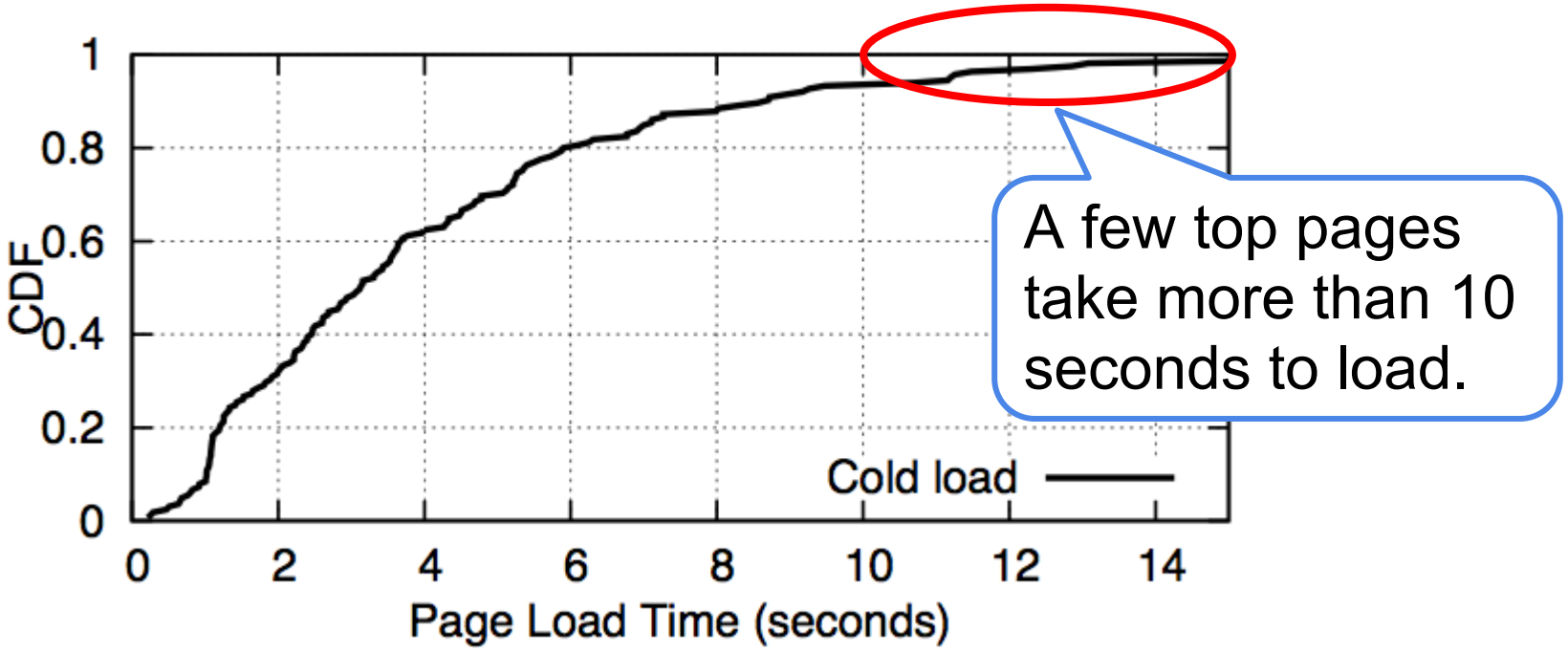
Page load is critical but slow

- Amazon can increase 1% revenue by decreasing page load time by 0.1s.
- Page load is slow even on top 200 websites



Page load is critical but slow

- Amazon can increase 1% revenue by decreasing page load time by 0.1s.
- Page load is slow even on top 200 websites



Many techniques aim to optimize page load time

- Optimization techniques
  - Server placement: CDNs
  - Web pages and cache: mod_pagespeed, Silo
  - Application level: SPDY
  - TCP/DNS: TCP fast open, ASAP, DNS pre-resolution, TCP pre-connect
- Problem
  - Unclear whether they help or hurt page loads*

*http://www.guypo.com/technical/not-as-spdy-as-you-thought/.

*http://www.stevesouders.com/

Many techniques aim to optimize page load time

- Optimization techniques
  - Server placement: CDNs
  - Web pages and cache: mod_pagespeed, Silo
  - Application level: SPDY
  - TCP/DNS: TCP fast open, ASAP, DNS pre-resolution, TCP pre-connect
- Problem
  - Unclear whether they help or hurt page loads*

*http://www.guypo.com/technical/not-as-spdy-as-you-thought/.
*http://www.stevesouders.com/

Page load process is poorly understood.

Difficult to understand page load

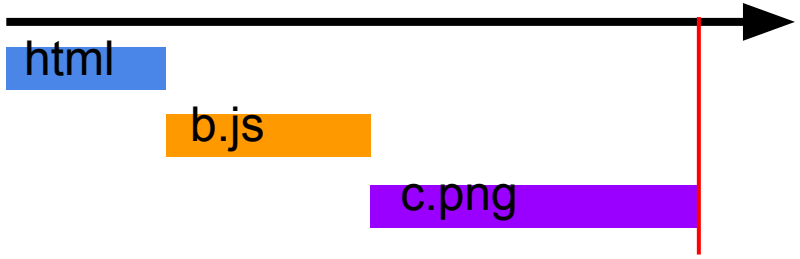
- Factors that affect page load
  - Page structure
  - Inter-dependencies between network and computation activities
  - Browser implementations

Difficult to understand page load

```
<html>
  <script src="b.js"></script>
  <img src="c.png"/>
</html>
```

```
<html>
  <img src="c.png"/>
  <script src="b.js"></script>
</html>
```

Difficult to understand page load

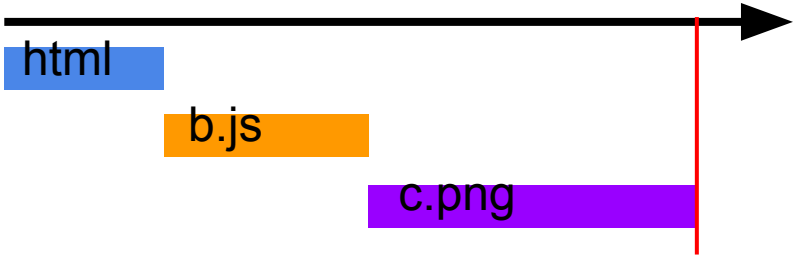



```
<html>
   <script src="b.js"></script>
   <img src="c.png"/>
</html>
```

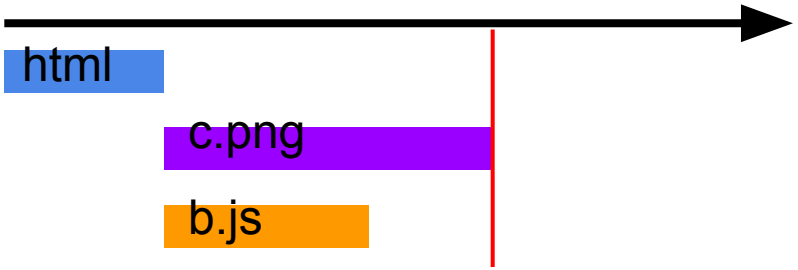
```
<html>
   <img src="c.png"/>
   <script src="b.js"></script>
</html>
```

Difficult to understand page load

```
<html>
   <script src="b.js"></script>
   <img src="c.png"/>
</html>
```

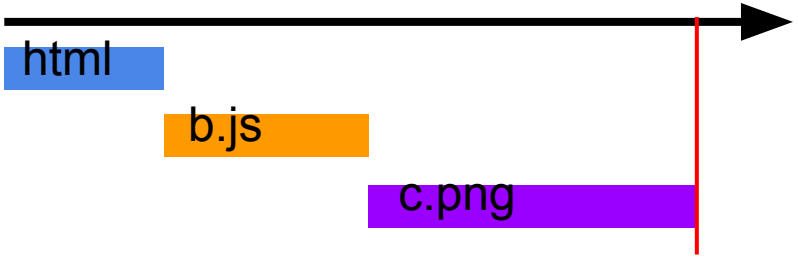



```
<html>
   <img src="c.png"/>
   <script src="b.js"></script>
</html>
```

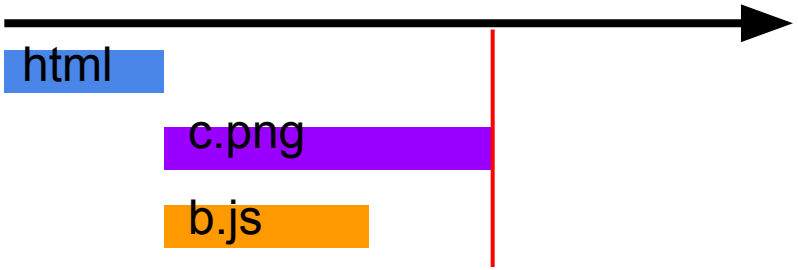


Difficult to understand page load

```
<html>
  <script src="b.js"></script>
  <img src="c.png"/>
</html>
```




```
<html>
  <img src="c.png"/>
  <script src="b.js"></script>
</html>
```



Understanding dependencies is the key to understand page load.

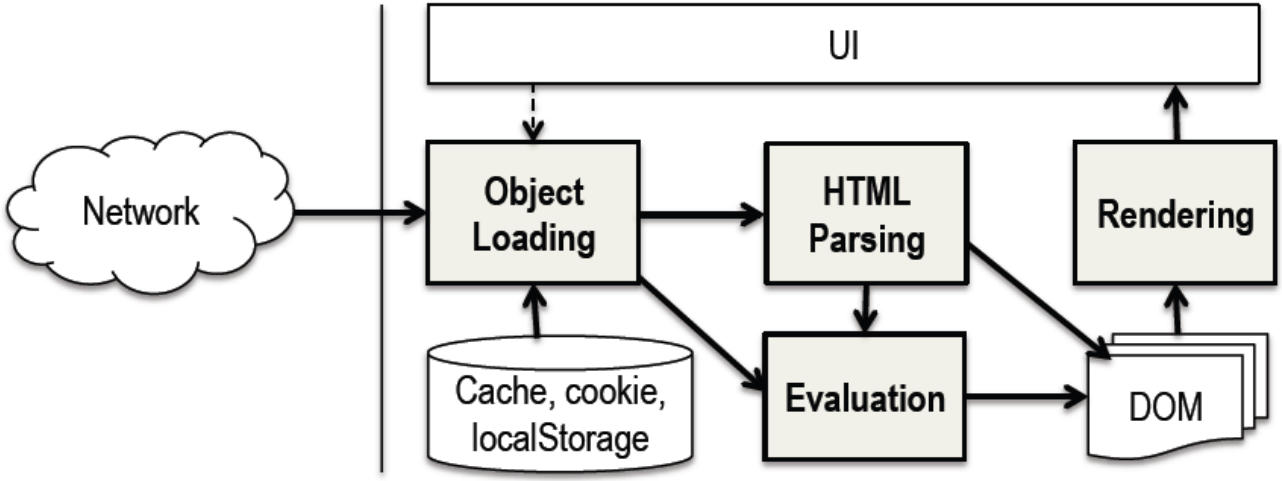
Overview of our work

- Model the page load process
- Build the WProf tool
- Study page load on real pages

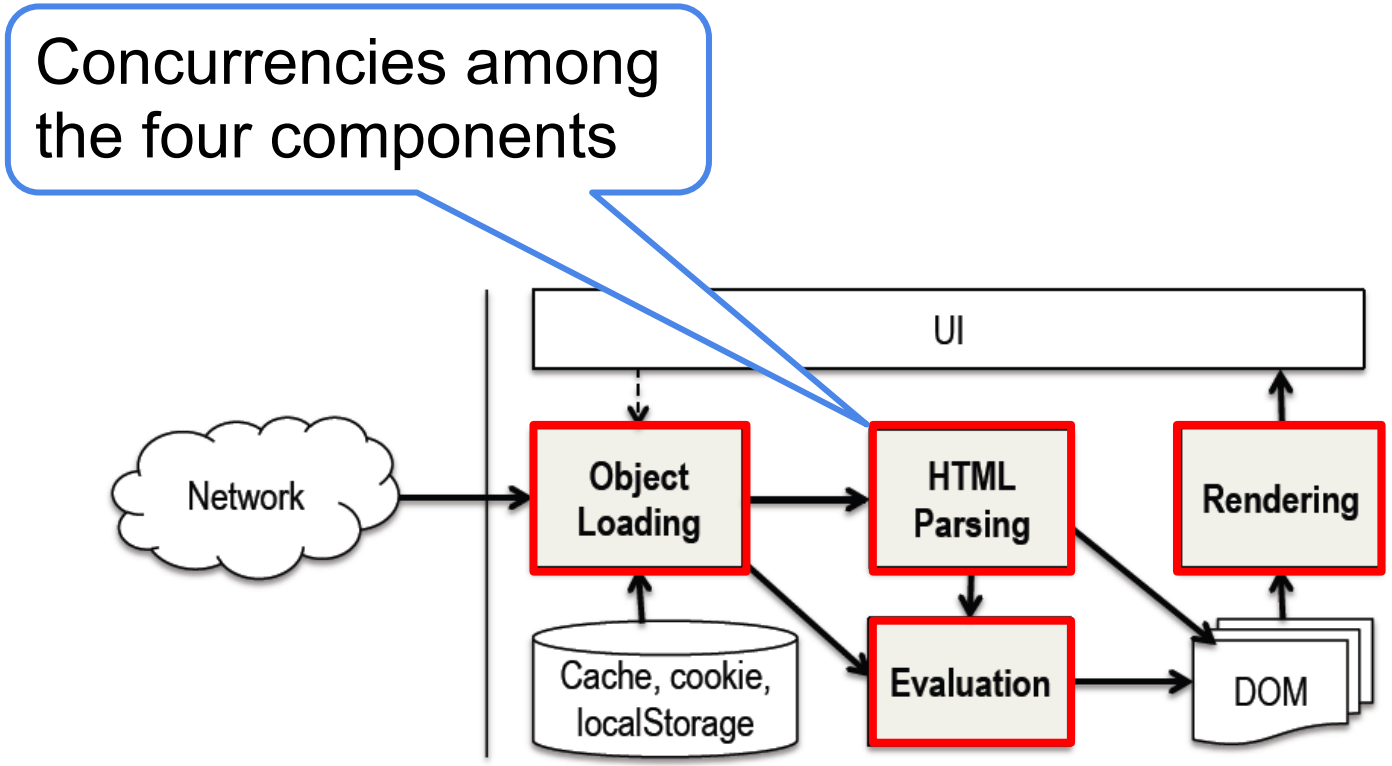
Overview of our work

- Model the page load process
  - How a page is loaded?
  - How to infer dependencies?
- Build the WProf tool
- Study page load on real pages

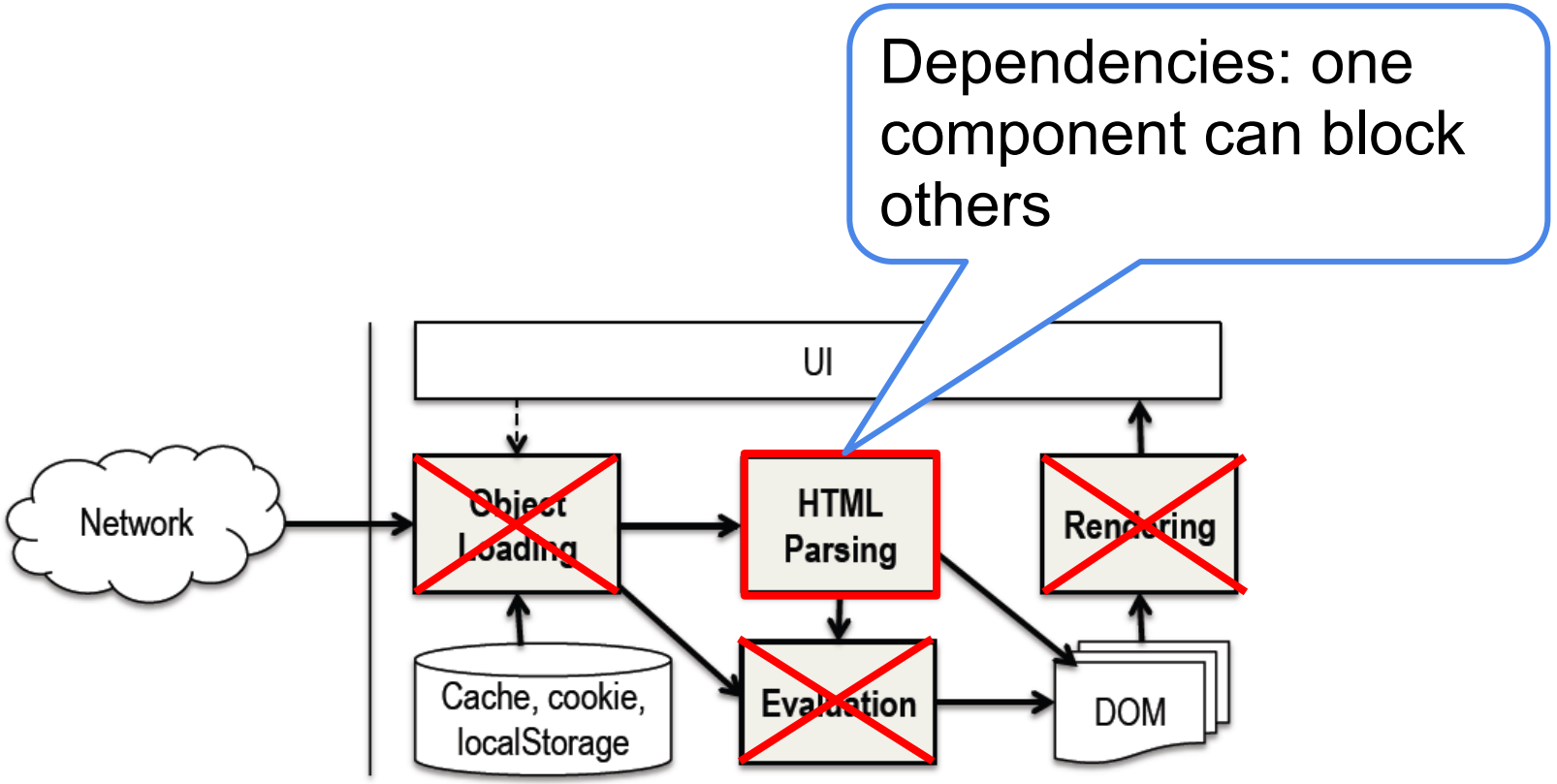
How a page is loaded



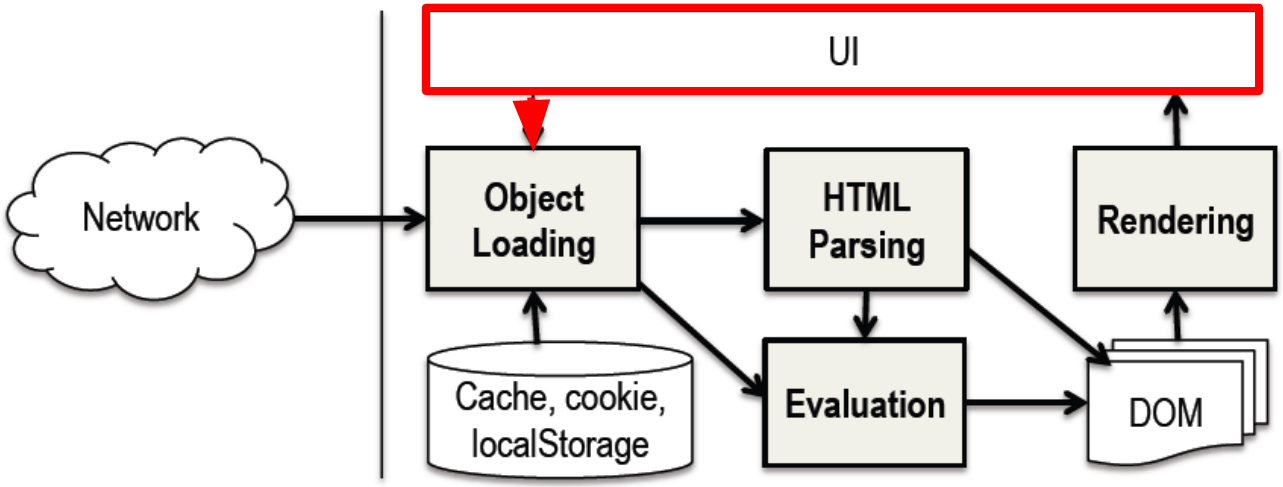
How a page is loaded



How a page is loaded



How a page is loaded

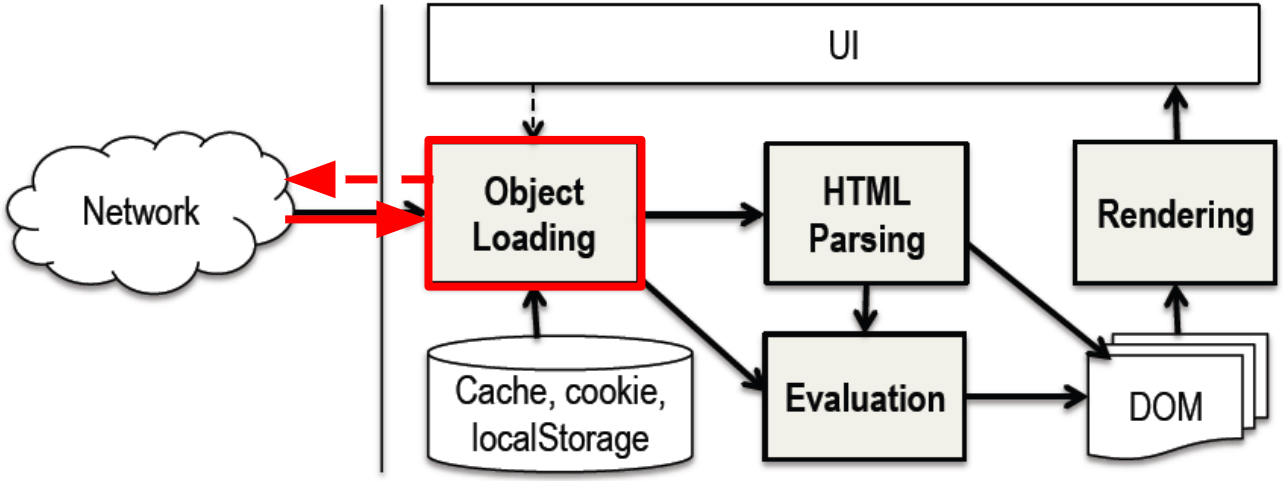


A page load starts with a user-initiated request.

How a page is loaded

index.html

```
1  <html>
2      <script src="main.js"/>
3  </html>
```

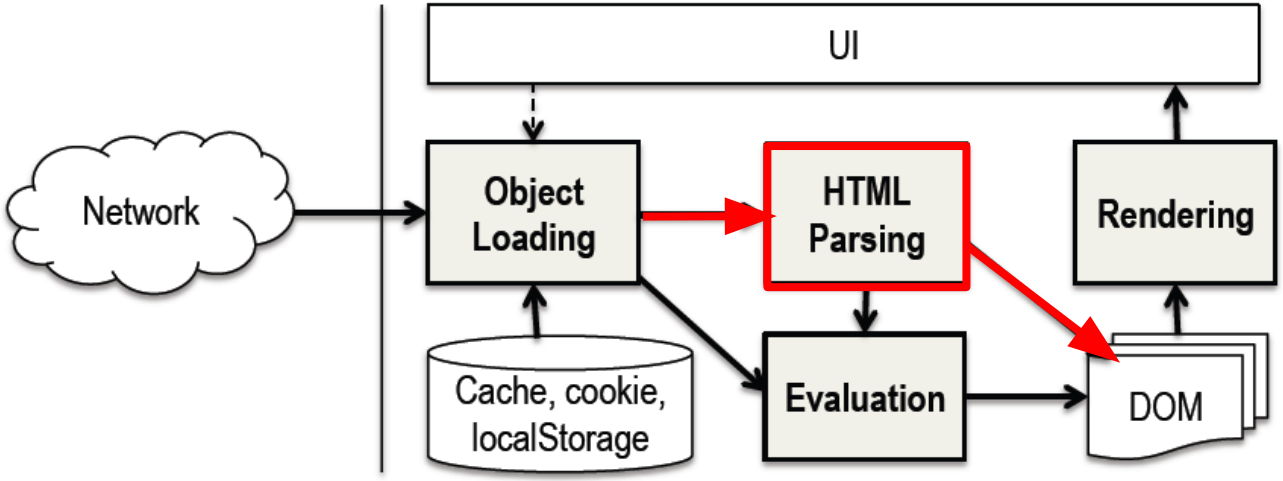


Object Loader downloads the corresponding Web page.

How a page is loaded

index.html

```
1  <html>
2      <script src="main.js"/>
3  </html>
```

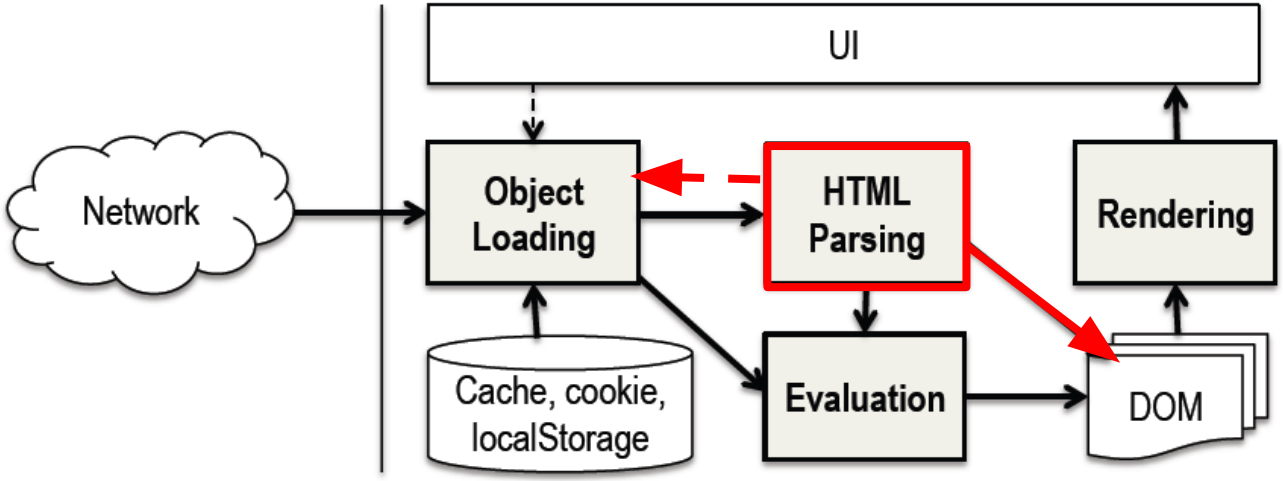


Upon receiving the first chunk of the root page, the HTML Parser starts to parse the page.

How a page is loaded

index.html

```
1  <html>
2  <script src="main.js"/>
3  </html>
```



HTML Parser requests embedded objects, i.e., JavaScript.

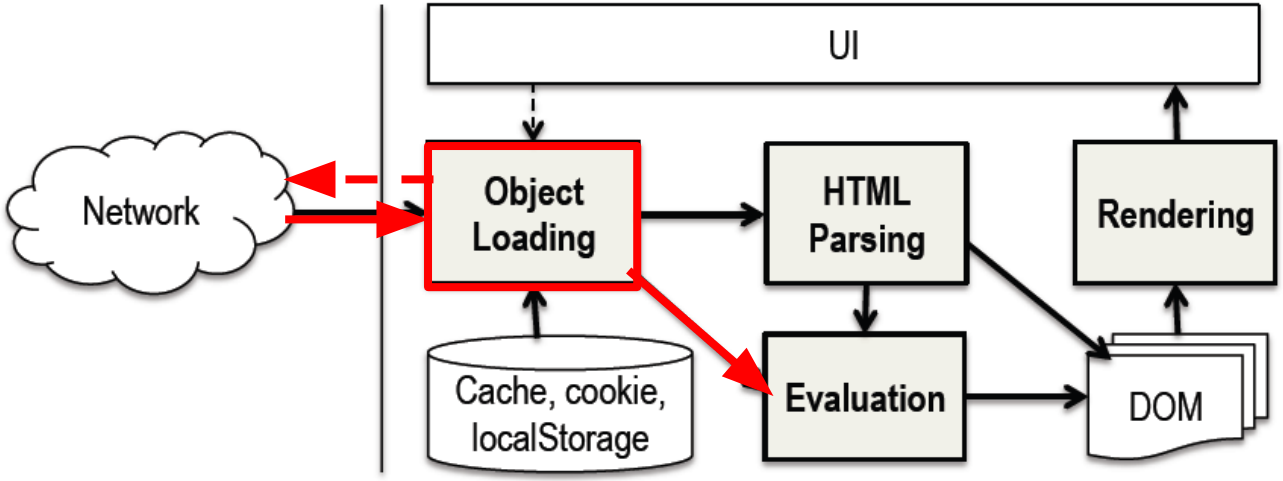
How a page is loaded

index.html

```
1  <html>
2  <script src="main.js"/>
3  </html>
```

main.js

```
...
```



Object Loader requests the inlined JS and sends it for evaluation.

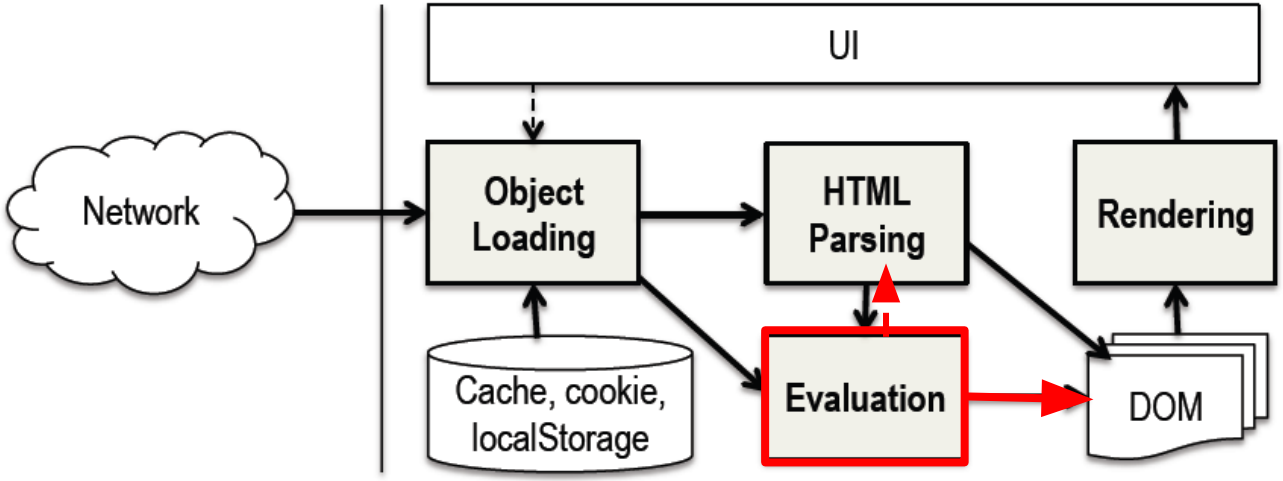
How a page is loaded

index.html

```
1  <html>
2  <script src="main.js"/>
3  </html>
```

main.js

```
...
```

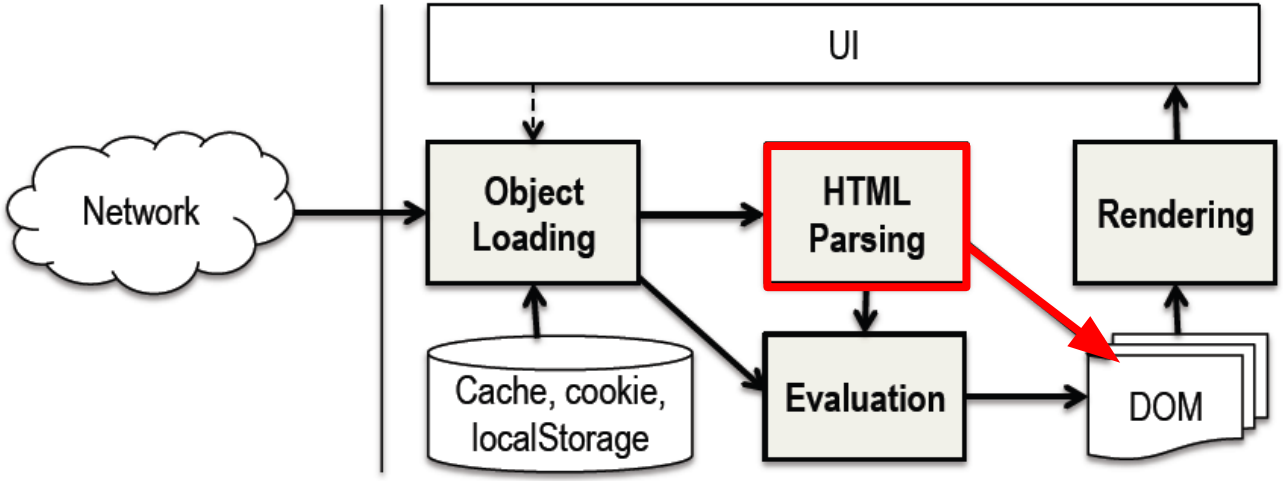


JS evaluation can modify the DOM and its completion resumes HTML parsing.

How a page is loaded

index.html

```
1  <html>
2      <script src="main.js"/>
3  </html>
```

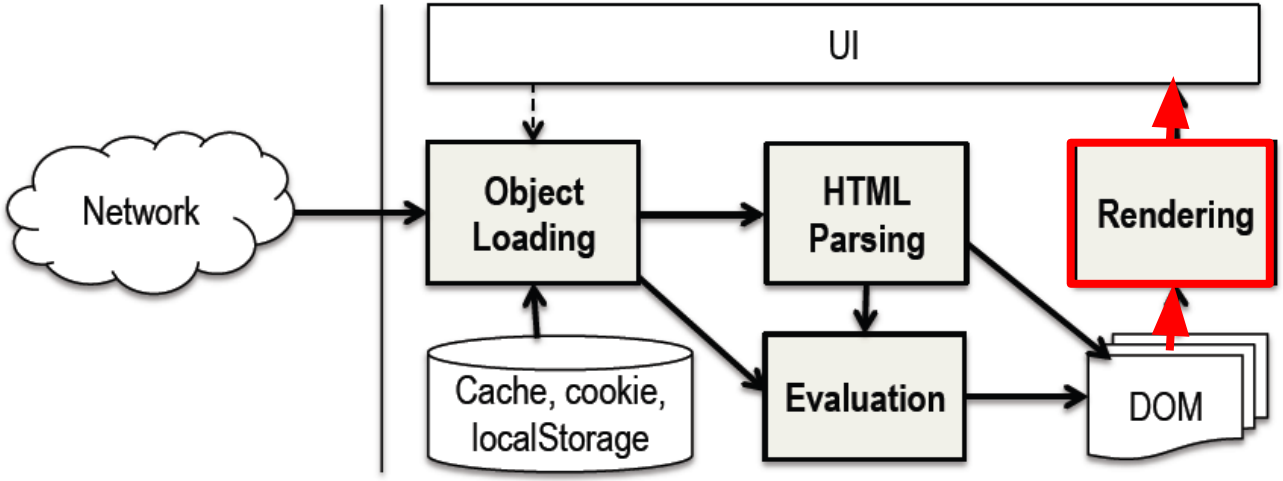


HTML continues being parsed and added to the DOM.

How a page is loaded

index.html

```
1  <html>
2      <script src="main.js"/>
3  </html>
```



Rendering Engine progressively renders the page (i.e., layout and painting).

How to infer dependencies

- Goal
  - Extract as many dependencies as possible across browsers
- Methodology
  - Design test pages
  - Examine documents
  - Inspect browser code

How to infer dependencies

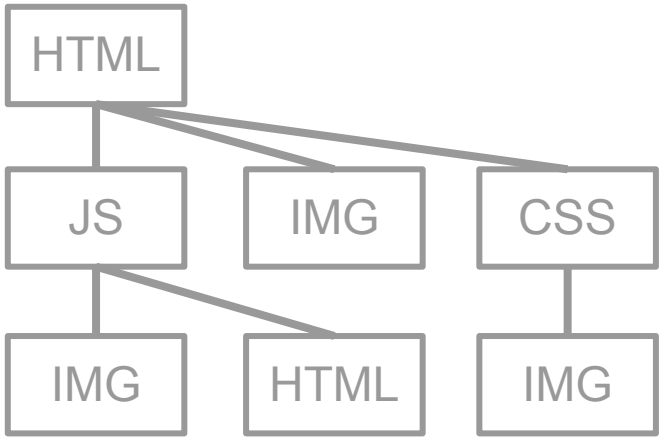
- Goal
  - Extract as many dependencies as possible across browsers
- Methodology
  - Design test pages
  - Examine documents
  - Inspect browser code

Reverse engineer page loads with test pages

- Design test pages

Reverse engineer page loads with test pages

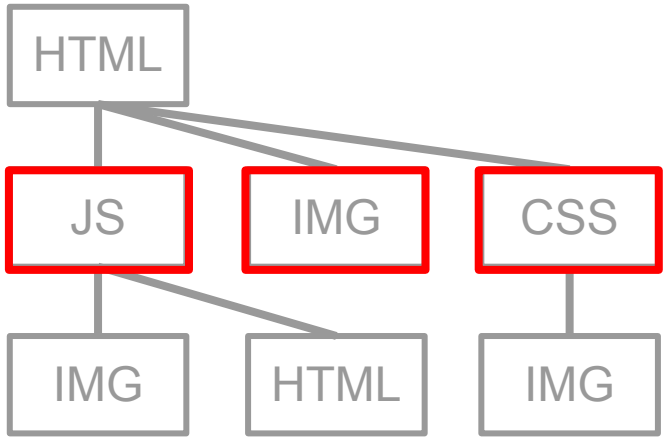
- Design test pages



An example Web page

Reverse engineer page loads with test pages

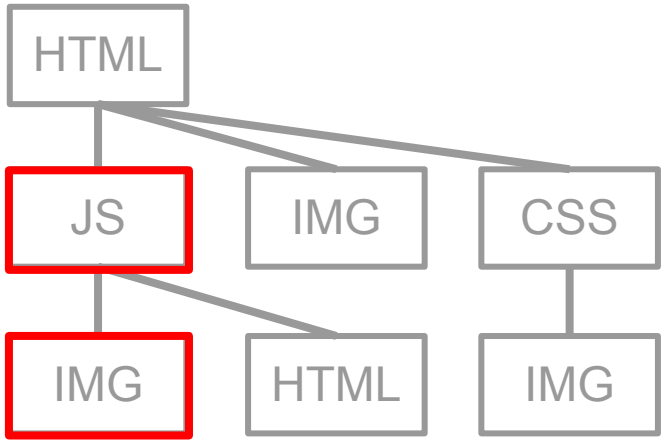
- Design test pages
  - An object follows another



An example Web page

Reverse engineer page loads with test pages

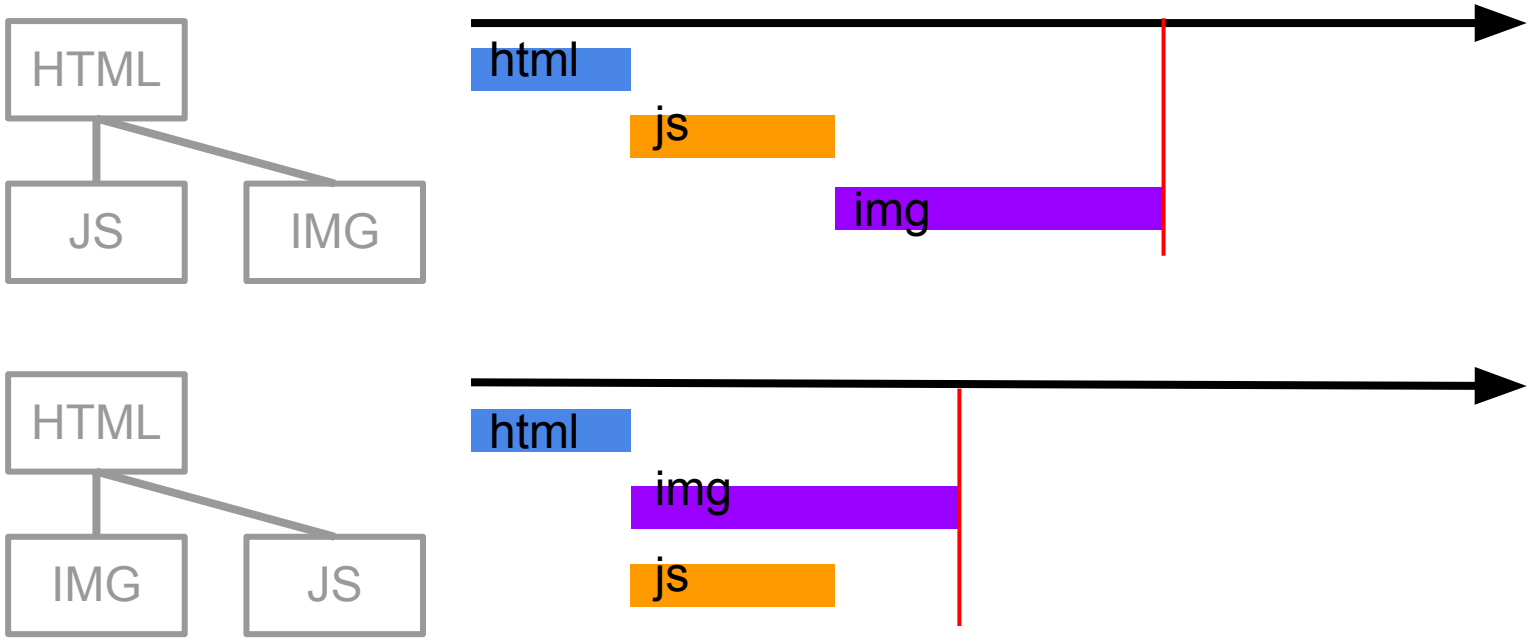
- Design test pages
  - An object follows another
  - An object embeds another



An example Web page

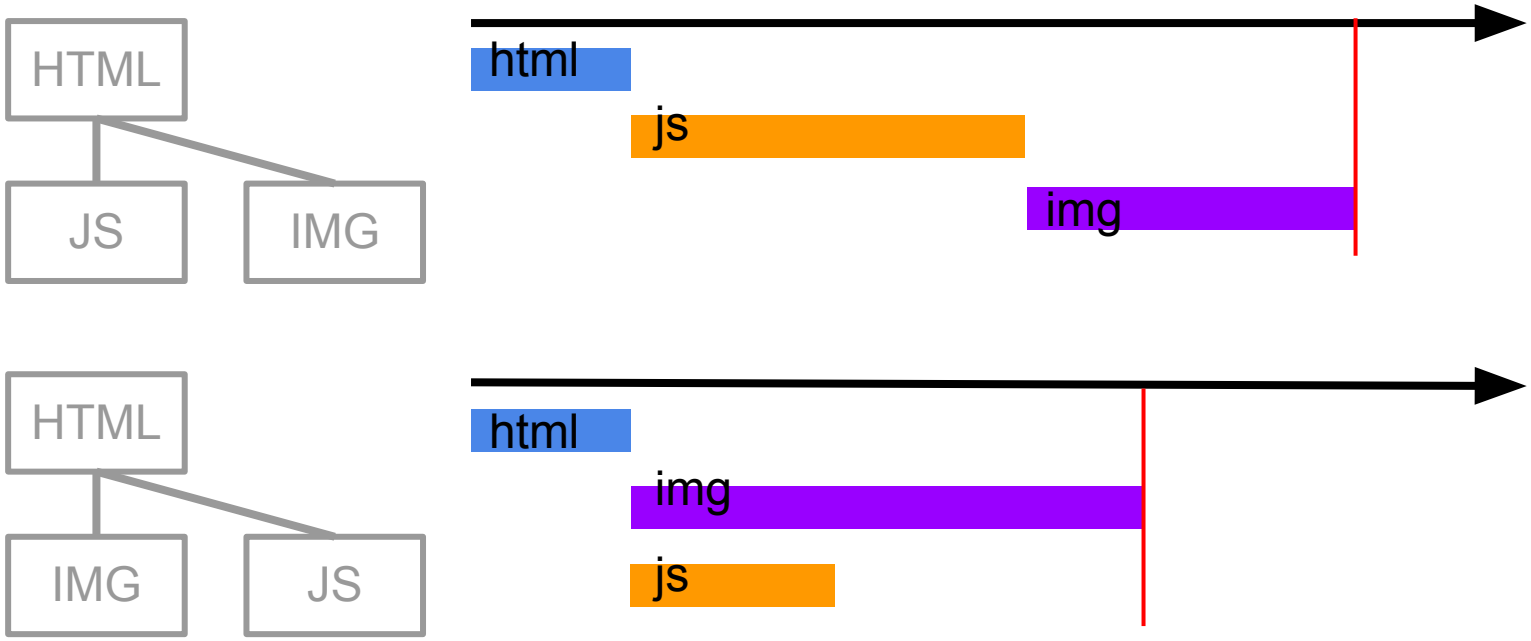
Reverse engineer page loads with test pages

- Design test pages
- Observe timings from DevTools



Reverse engineer page loads with test pages

- Design test pages
- Observe timings from DevTools



Dependency policy categories

- Flow dependency
  - Natural order that activities occur

Dependency policy categories

- Flow dependency
- Output dependency
  - Correctness of execution when multiple processes access to the same resource

Dependency policy categories

- Flow dependency
- Output dependency
- Lazy/Eager binding
  - Tradeoffs between data downloads and page load latencies

Dependency policy categories

- Flow dependency
- Output dependency
- Lazy/Eager binding
- Resource constraints
  - Limited computing power or network resources (# TCP conn.)

Output dependency

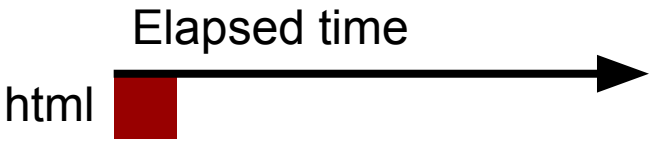
index.html

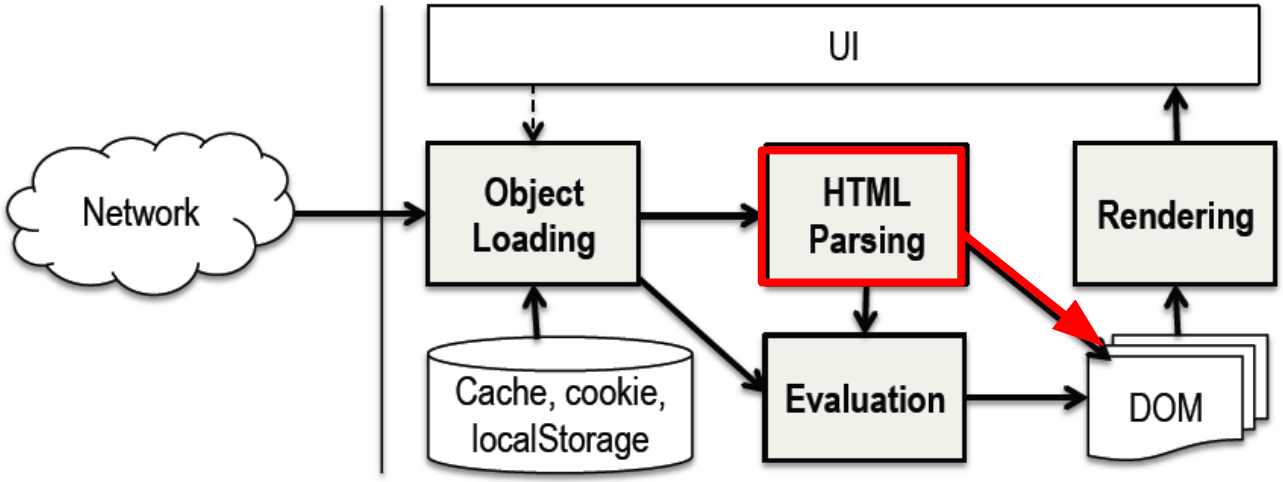
```
1  <html>
2      <link rel="stylesheet" href="c.css">
3      <script src="f.js"/>
   ...
```

Output dependency

index.html

```
1  <html>
2      <link rel="stylesheet" href="c.css">
3      <script src="f.js"/>
    ...
```

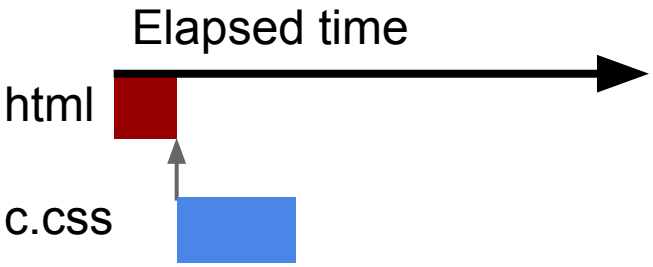


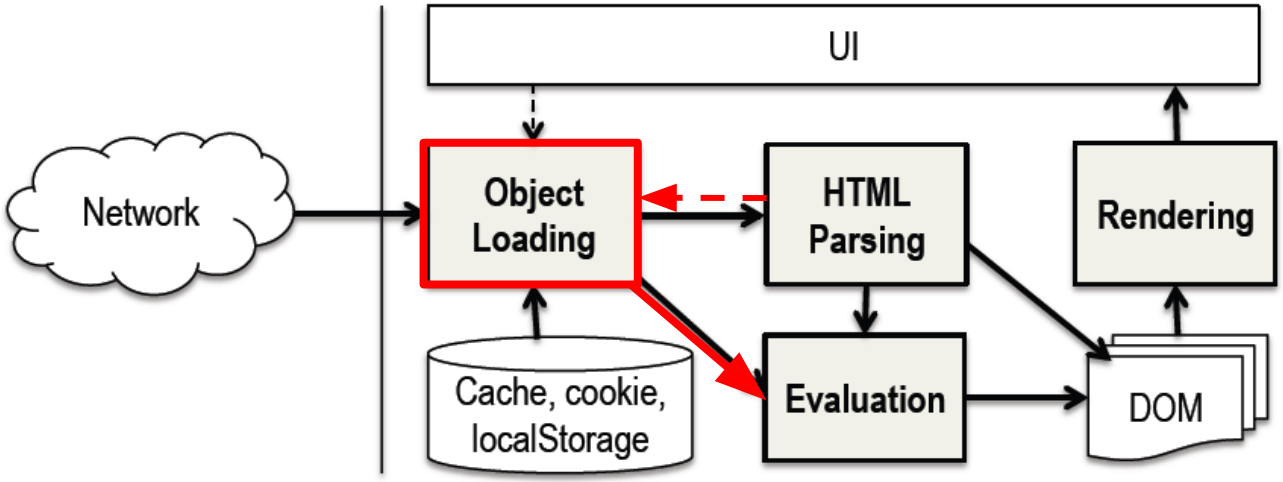


Output dependency

index.html

```
1  <html>
2      <link rel="stylesheet" href="c.css">
3      <script src="f.js"/>
    ...
```

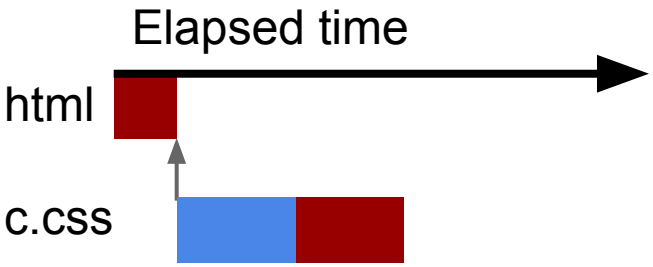


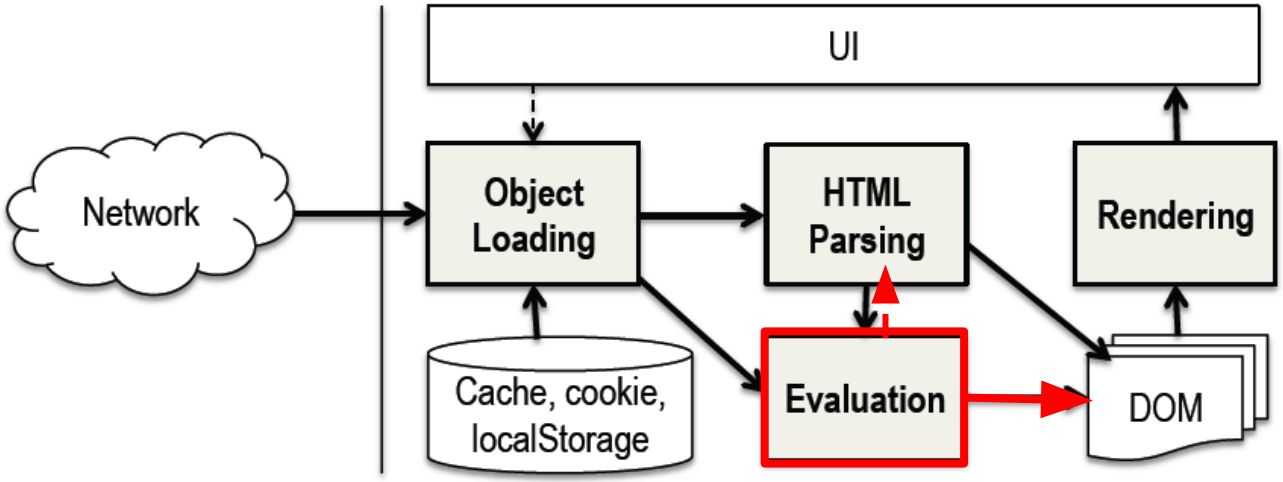


Output dependency

index.html

```
1  <html>
2      <link rel="stylesheet" href="c.css">
3      <script src="f.js"/>
    ...
```

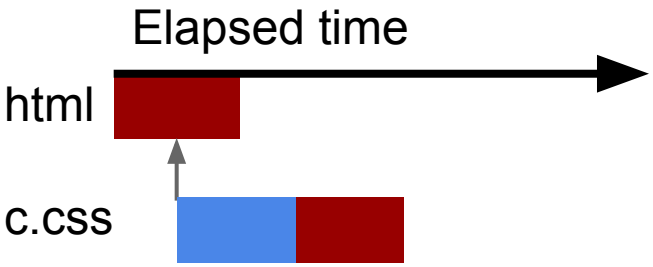


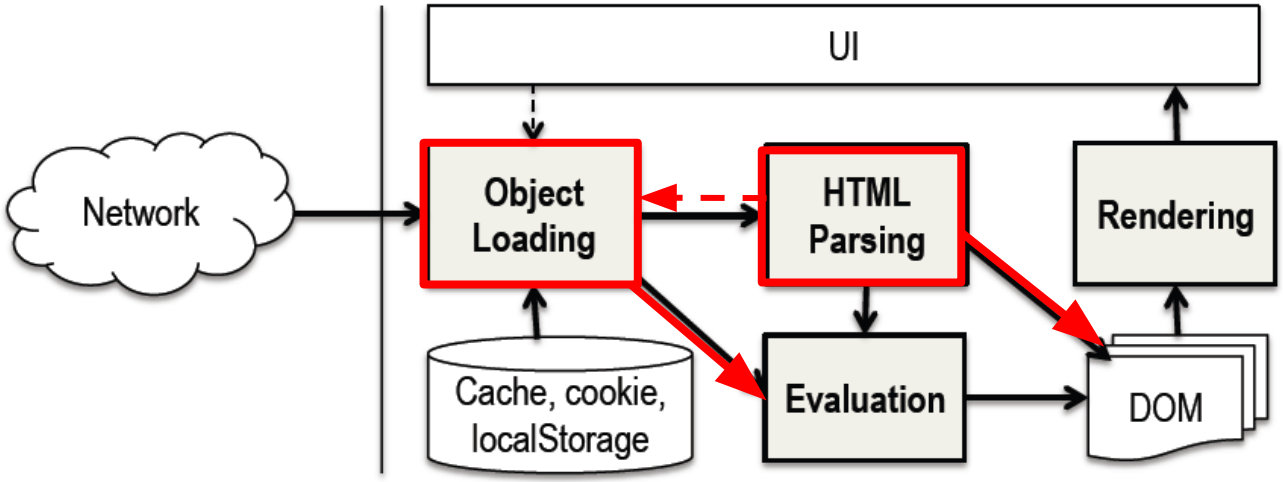


Output dependency

index.html

```
1  <html>
2      <link rel="stylesheet" href="c.css">
3      <script src="f.js"/>
    ...
```

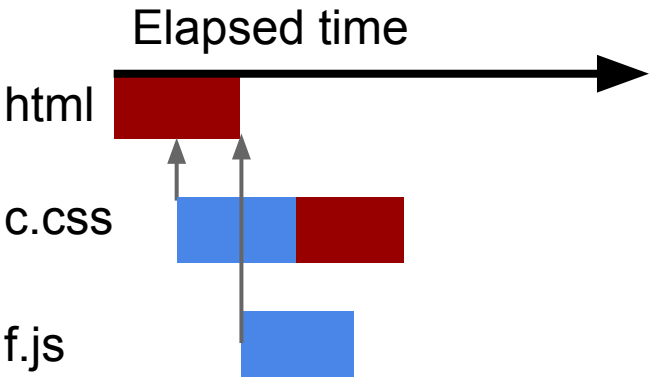


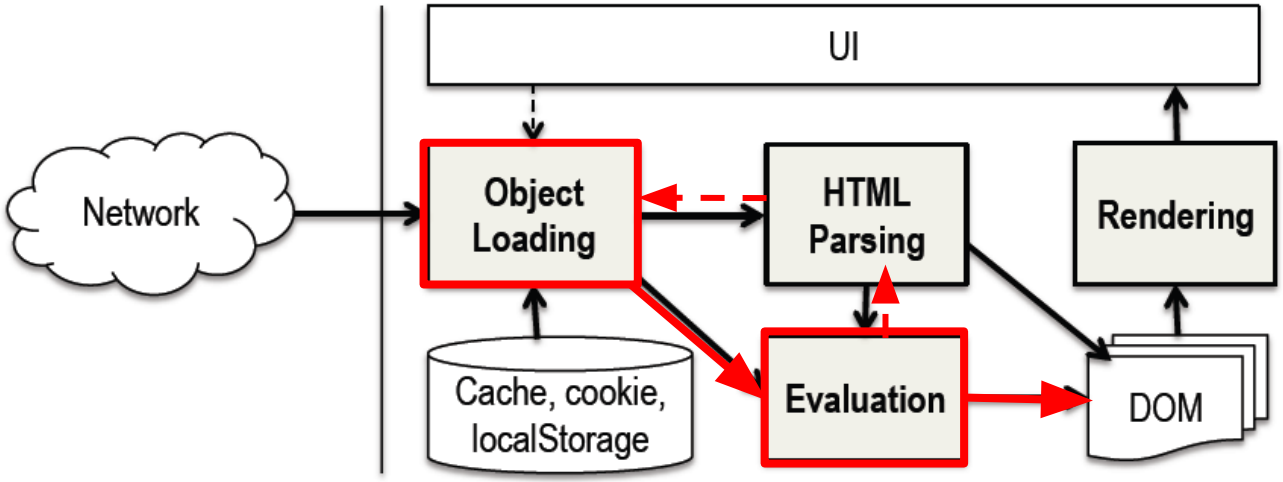


Output dependency

index.html

```
1  <html>
2      <link rel="stylesheet" href="c.css">
3      <script src="f.js"/>
    ...
```

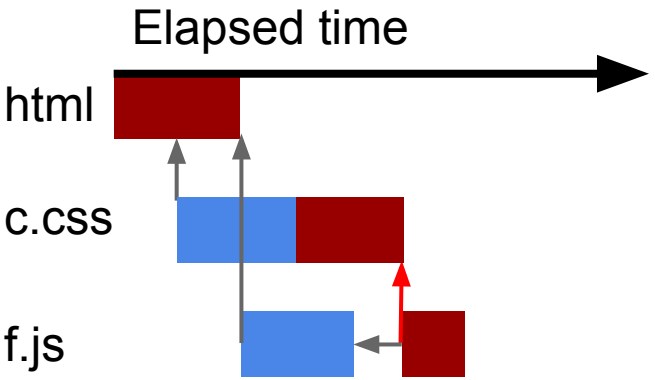


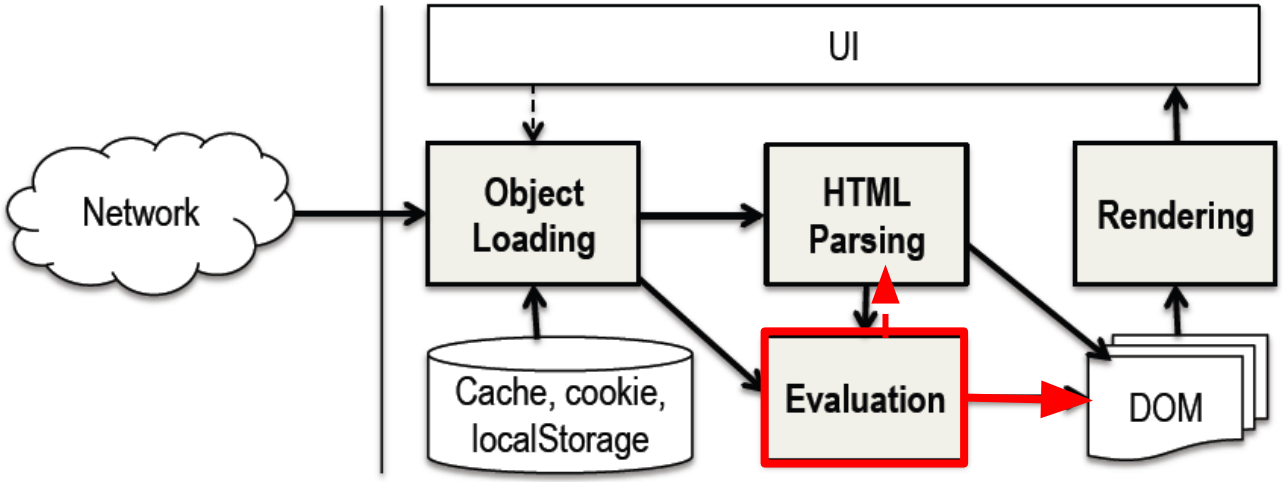


Output dependency

index.html

```
1  <html>
2      <link rel="stylesheet" href="c.css">
3      <script src="f.js"/>
     ...
```

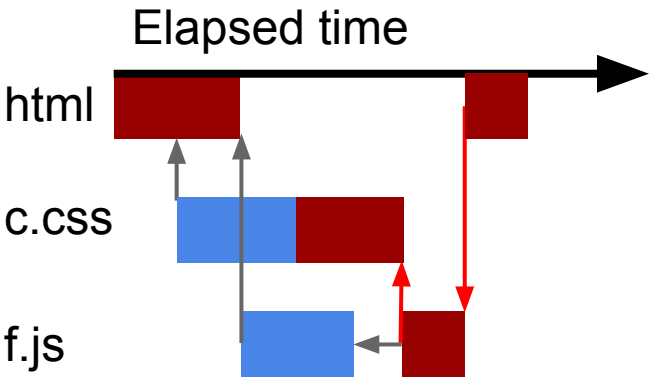


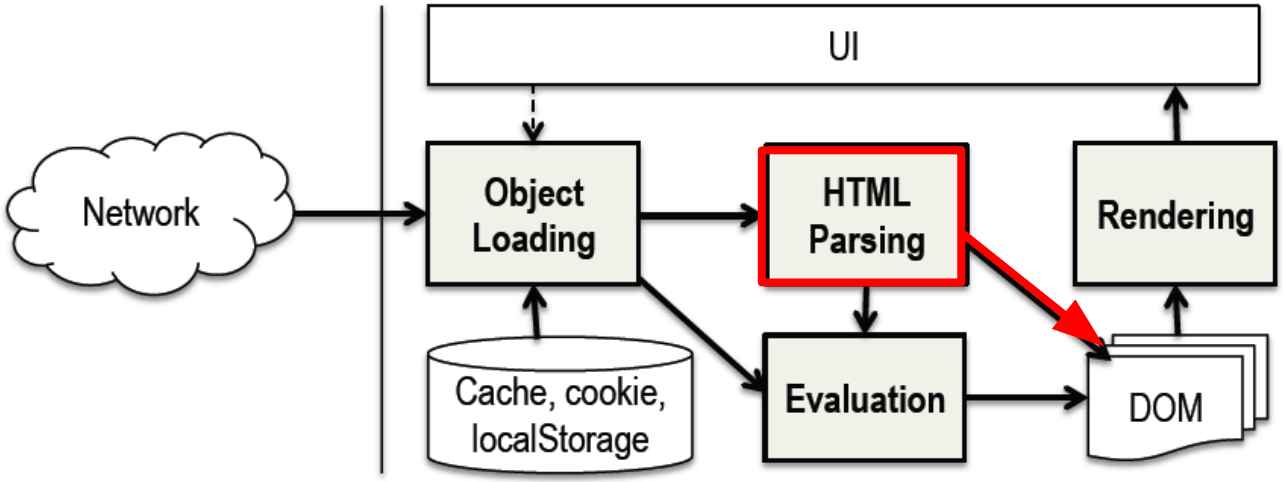


Output dependency

index.html

```
1  <html>
2      <link rel="stylesheet" href="c.css">
3      <script src="f.js"/>
     ...
```





Dependency policies

Dependency	Name	Definition
Flow	F1	Loading an object → Parsing the tag that references the object
	F2	Evaluating an object → Loading the object
	F3	Parsing the HTML page → Loading the first block of the HTML page*
	F4	Rendering the DOM tree → Updating the DOM
	F5	Loading an object referenced by a JavaScript or CSS → Evaluating the JavaScript or CSS*
	F6	Downloading/Evaluating an object → Listener triggers or timers
Output	O1	Parsing the next tag → Completion of a previous JavaScript download and evaluation
	O2	JavaScript evaluation → Completion of a previous CSS evaluation
	O3	Parsing the next tag → Completion of a previous CSS download and evaluation
Lazy/Eager binding	B1	[Lazy] Loading an image appeared in a CSS → Parsing the tag decorated by the image
	B2	[Lazy] Loading an image appeared in a CSS → Evaluation of any CSS that appears in front of the tag decorated by the image
	B3	[Eager] Preloading embedded objects does not depend on the status of HTML parsing. (breaks F1)
Resource constraint	R1	Number of objects fetched from different servers → Number of TCP connections allowed per domain
	R2	Browsers may execute key computational activities on the same thread, creating dependencies among the activities.This dependency is determined by the scheduling policy.

* An activity depends on *partial* completion of another activity.

Dependency policies

Dependency	Name	Definition
Flow	F1	Loading an object → Parsing the tag that references the object
	F2	Evaluating an object → Loading the object
	F3	Parsing the HTML page → Loading the first block of the HTML page*
	F4	Rendering the DOM tree → Updating the DOM
Output	F5	Loading an object referenced by a JavaScript or CSS → Evaluating the JavaScript or CSS*
	F6	Downloading/Evaluating an object → Listener triggers or timers
Output	O1	Parsing the next tag → Completion of a previous JavaScript download and evaluation
	O2	JavaScript evaluation → Completion of a previous CSS evaluation
	O3	Parsing the next tag → Completion of a previous CSS download and evaluation
Lazy/Eager binding	B1	[Lazy] Loading an image appeared in a CSS → Parsing the tag decorated by the image
	B2	[Lazy] Loading an image appeared in a CSS → Evaluation of any CSS that appears in front of the tag decorated by the image
	B3	[Eager] Preloading embedded objects does not depend on the status of HTML parsing. (breaks F1)
Resource constraint	R1	Number of objects fetched from different servers → Number of TCP connections allowed per domain
	R2	Browsers may execute key computational activities on the same thread, creating dependencies among the activities.This dependency is determined by the scheduling policy.

* An activity depends on *partial* completion of another activity

Flow 6

Output 3

Lazy/Eager binding 3

Resource constraint 2

Dependency policies across browsers

Dependency	IE	Firefox	WebKit
Output	all	no O3	no O3
Late binding	all	all	all
Eager binding	Preloads img, js, css	Preloads img, js, css	Preloads css, js
Net resource	6 conn.	6 conn.	6 conn.

Dependency policies across browsers

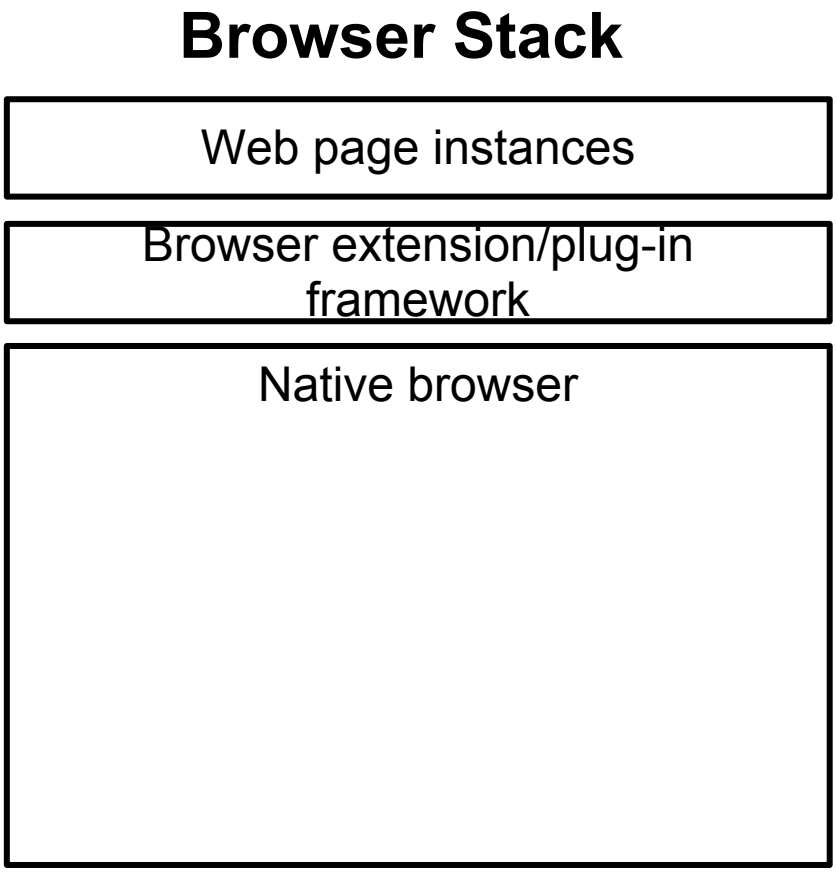
Dependency	IE	Firefox	WebKit
Output	all	no O3	no O3
Late binding	all	all	all
Eager binding	Preloads img, js, css	Preloads img, js, css	Preloads css, js
Net resource	6 conn.	6 conn.	6 conn.

O3: CSS downloads and evaluation block HTML parsing.

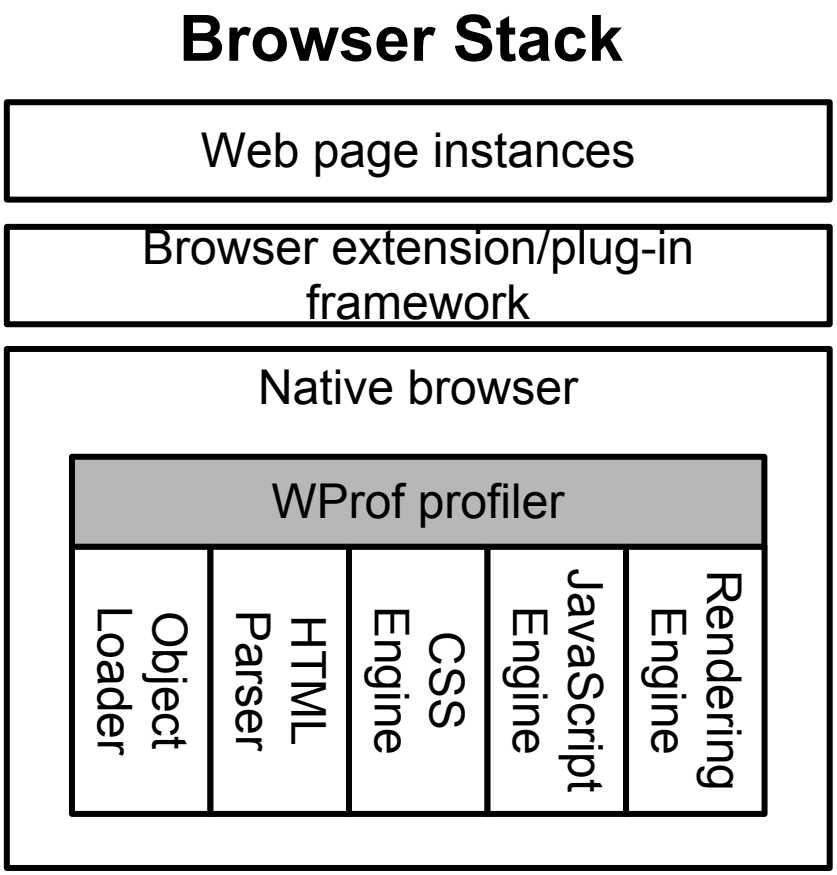
Overview of our work

- Model the page load process
- Build the WProf tool
  - Profiling in browsers
  - Generating dependency graphs
  - Analyzing critical paths
- Study page load on real pages

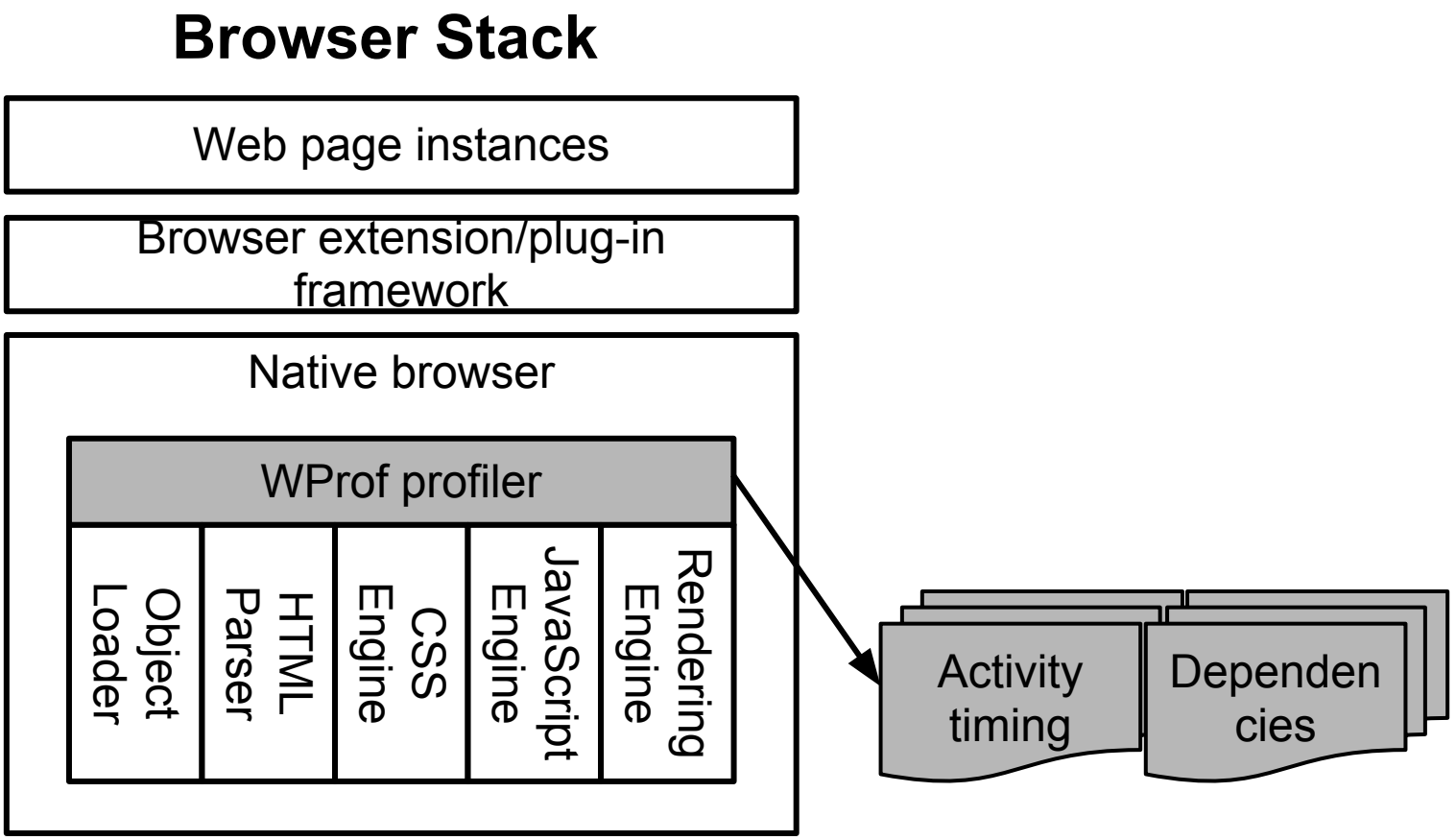
WProf architecture



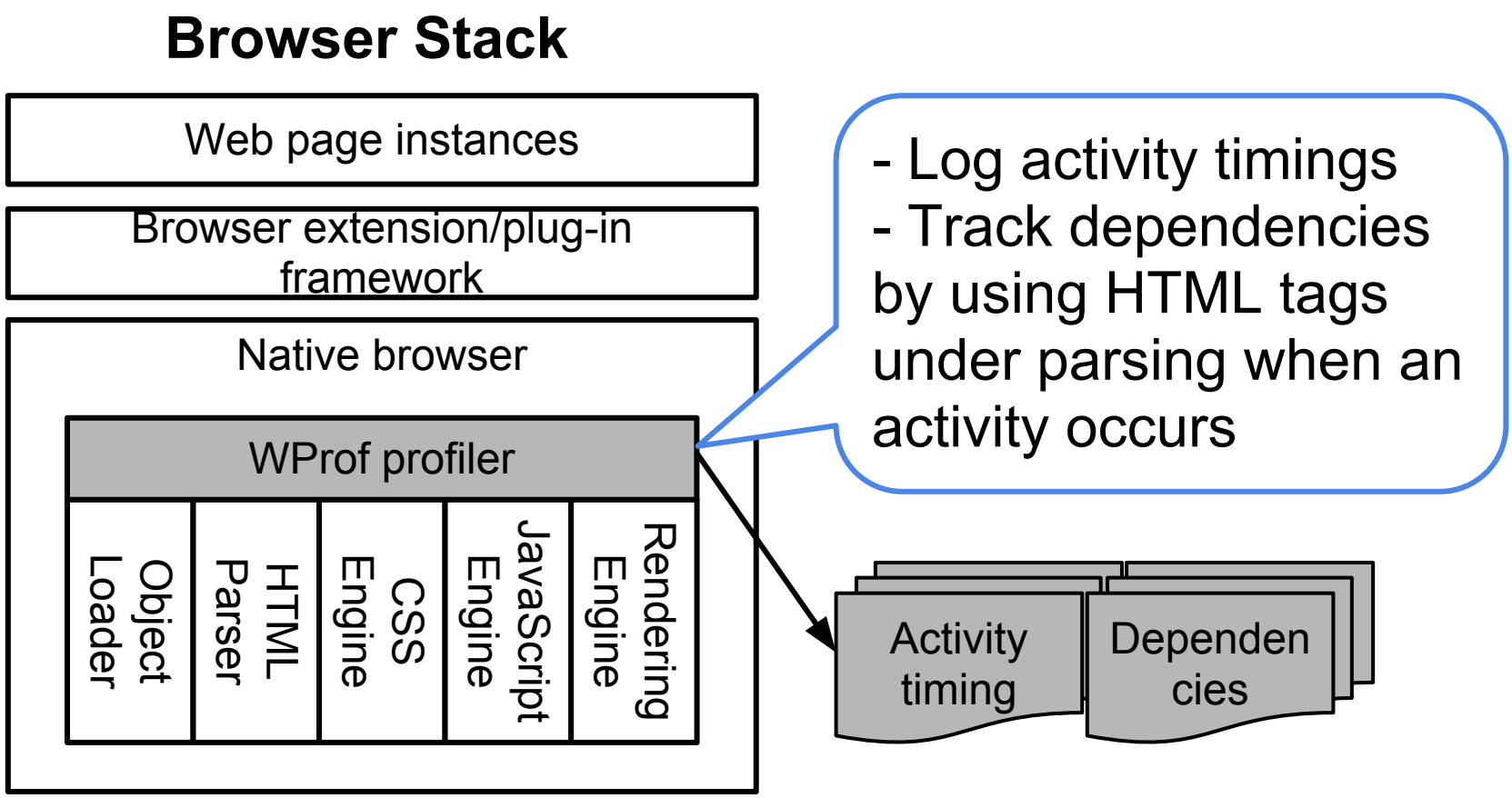
WProf architecture



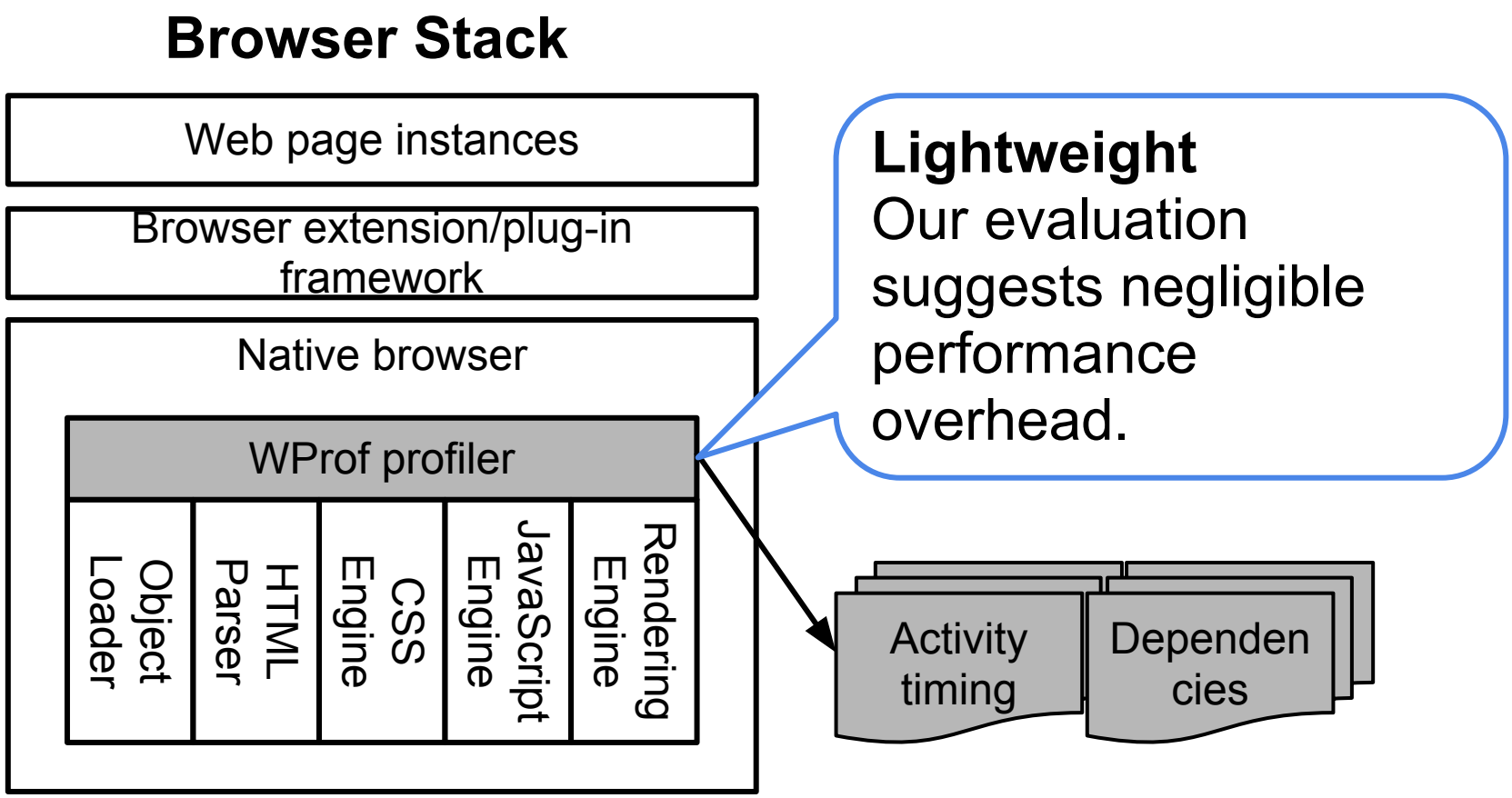
WProf architecture



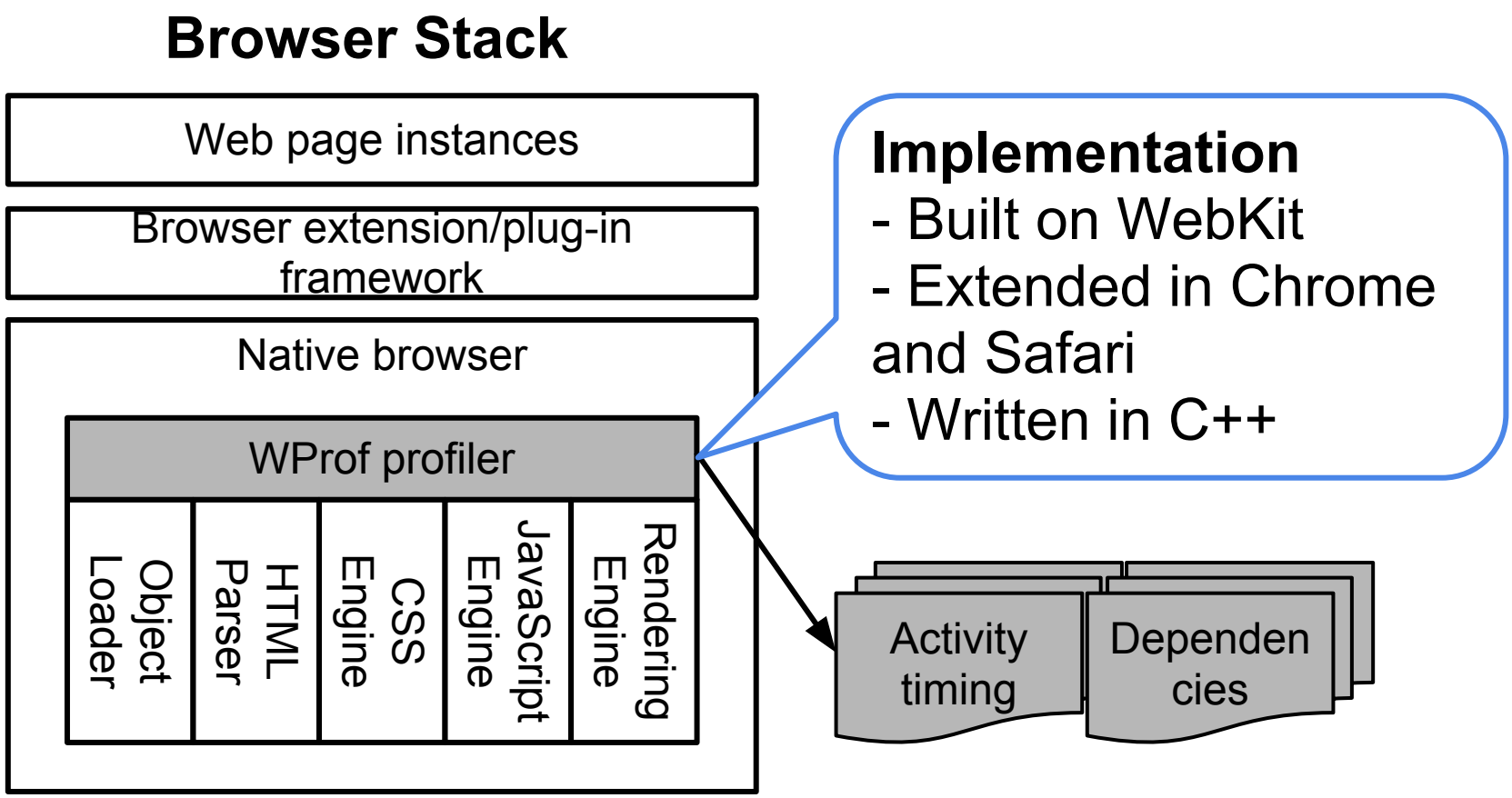
WProf architecture



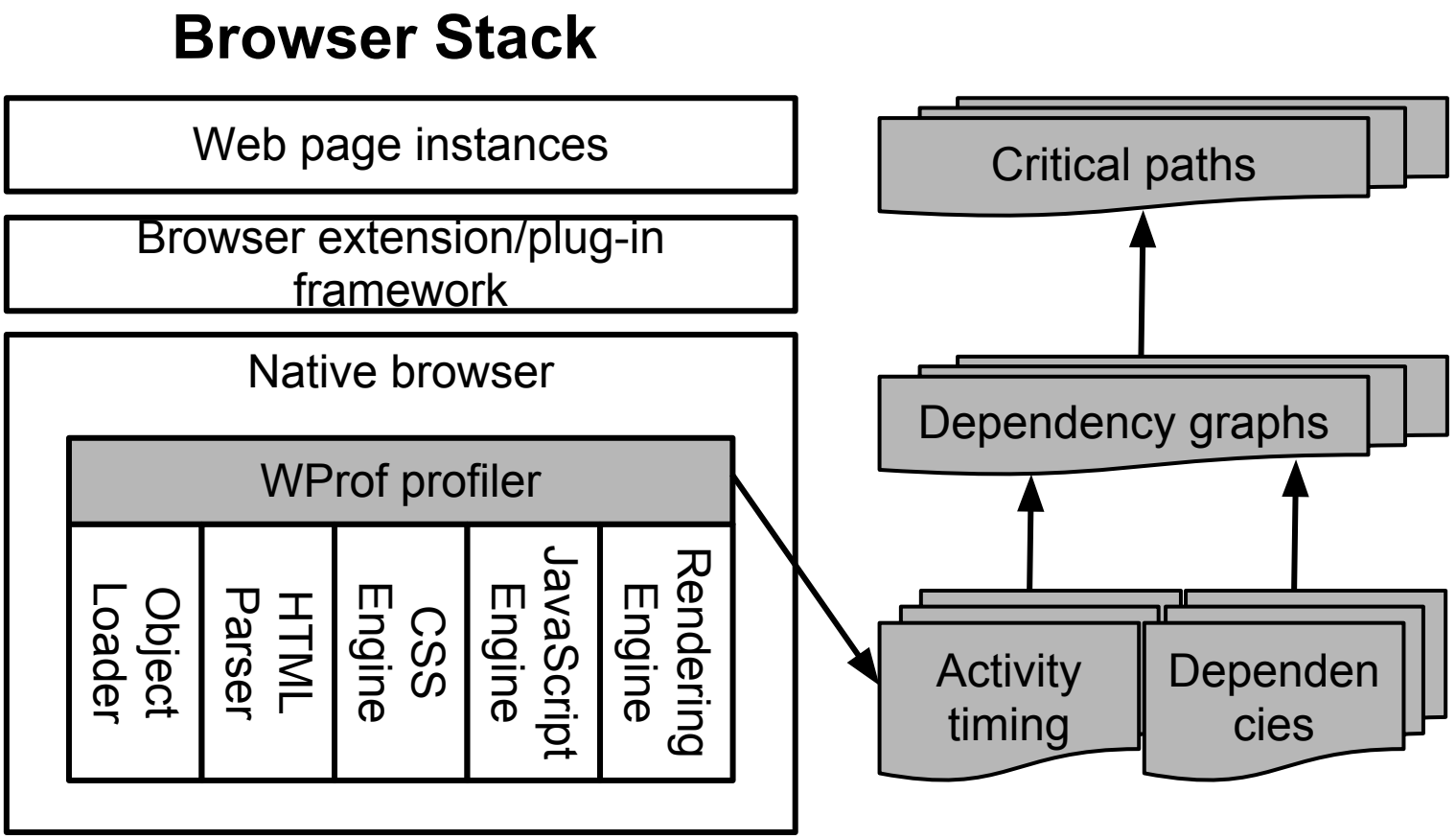
WProf architecture



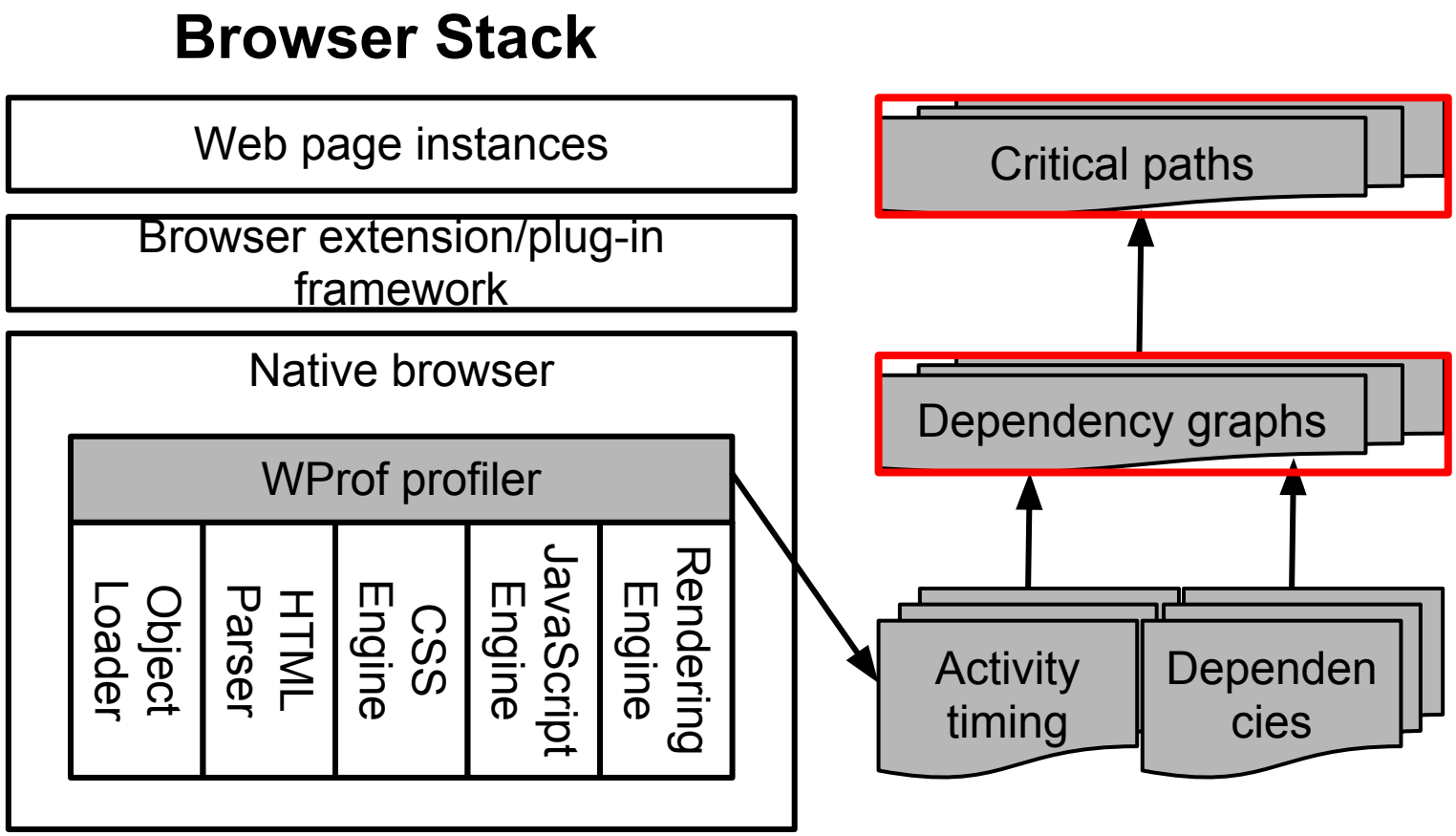
WProf architecture



WProf architecture

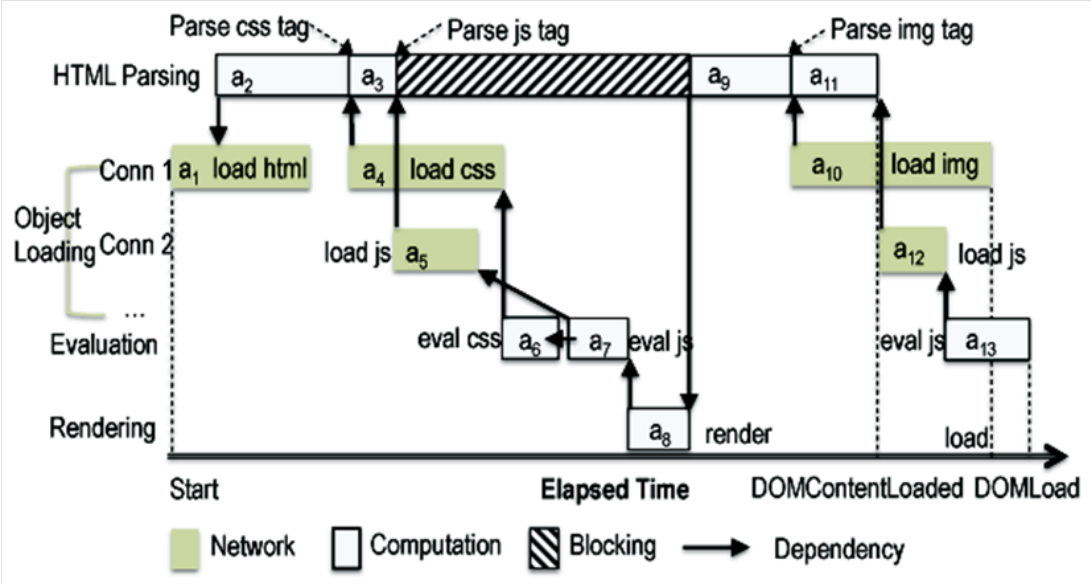


WProf architecture



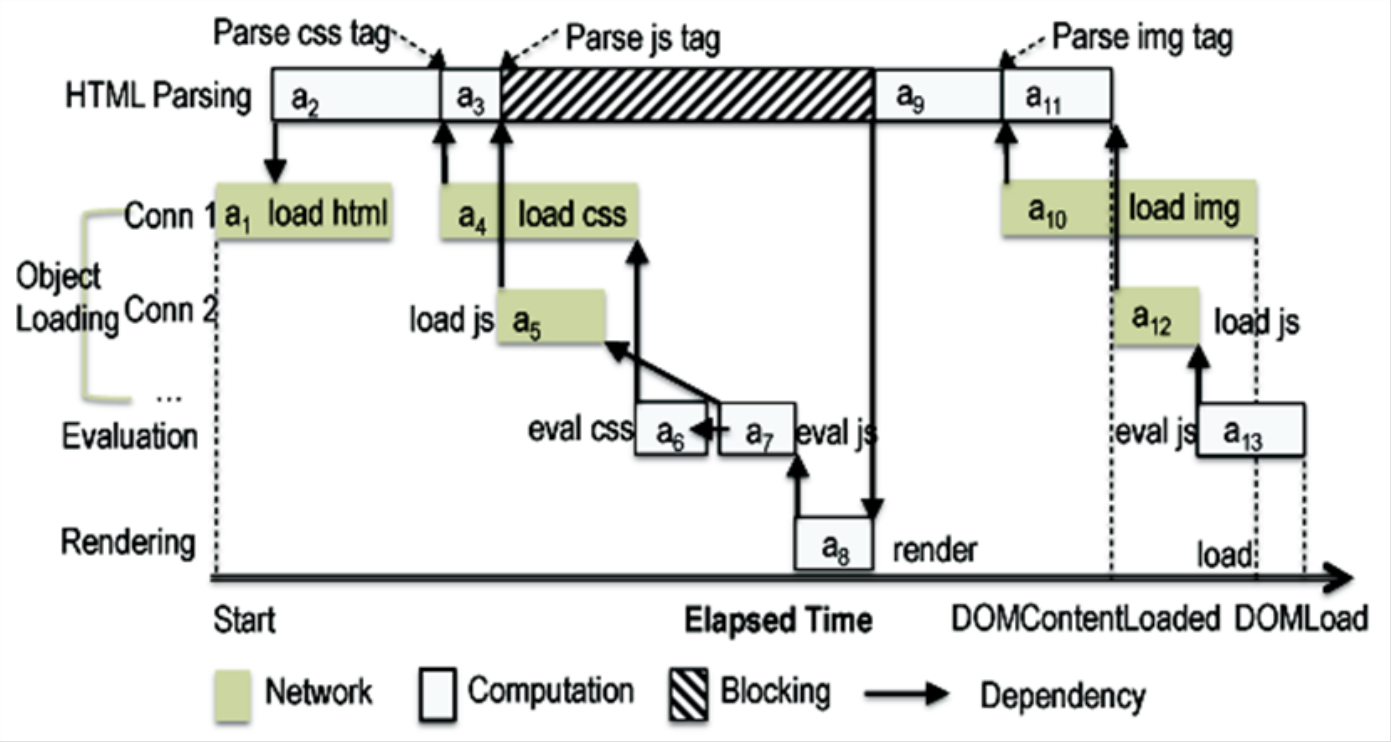
Dependency graph

```
<html>
   <head>
      <link rel="stylesheet" src="./main.css">
      <script src="./main.js" />
   </head>
   <!--request a JS-->
   <body onload="...">
      <img src="test.png" />
   </body>
</html>
```



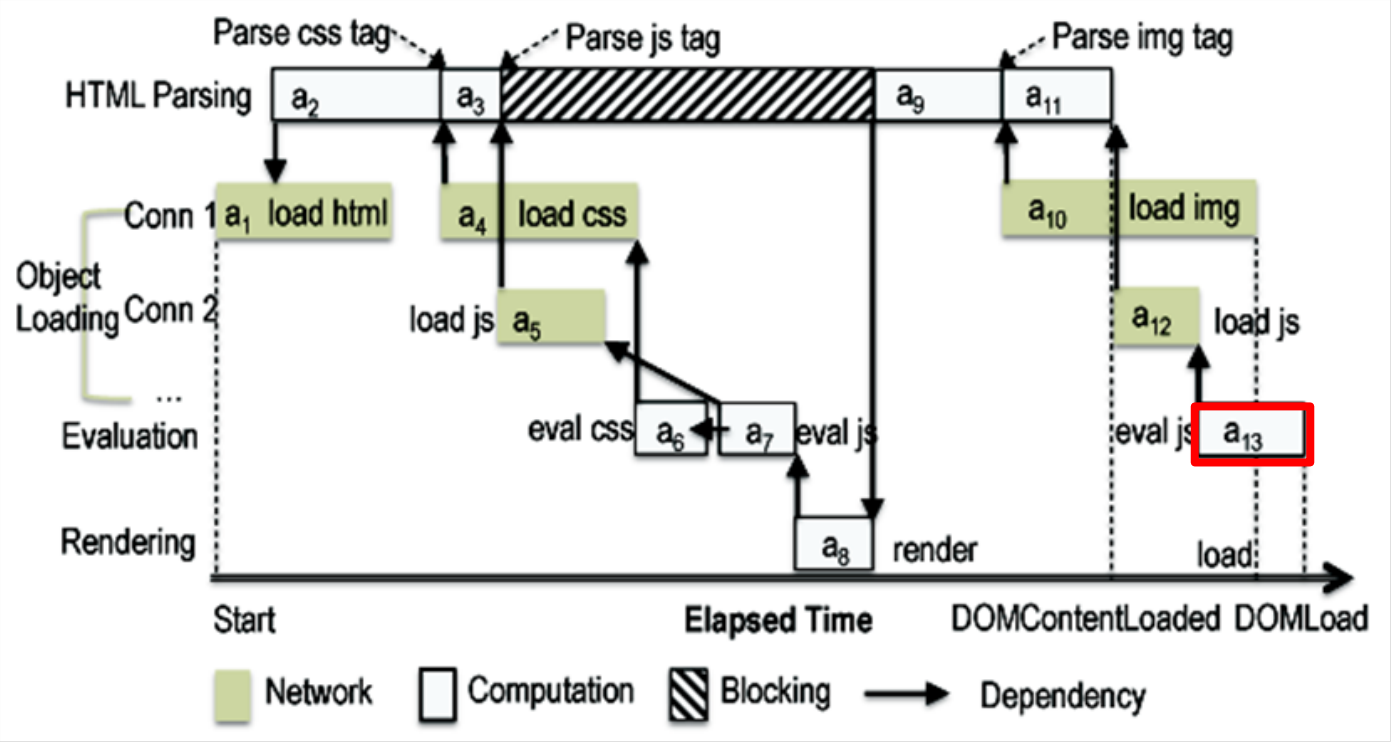
Critical path analysis

Critical path: the longest bottleneck path.



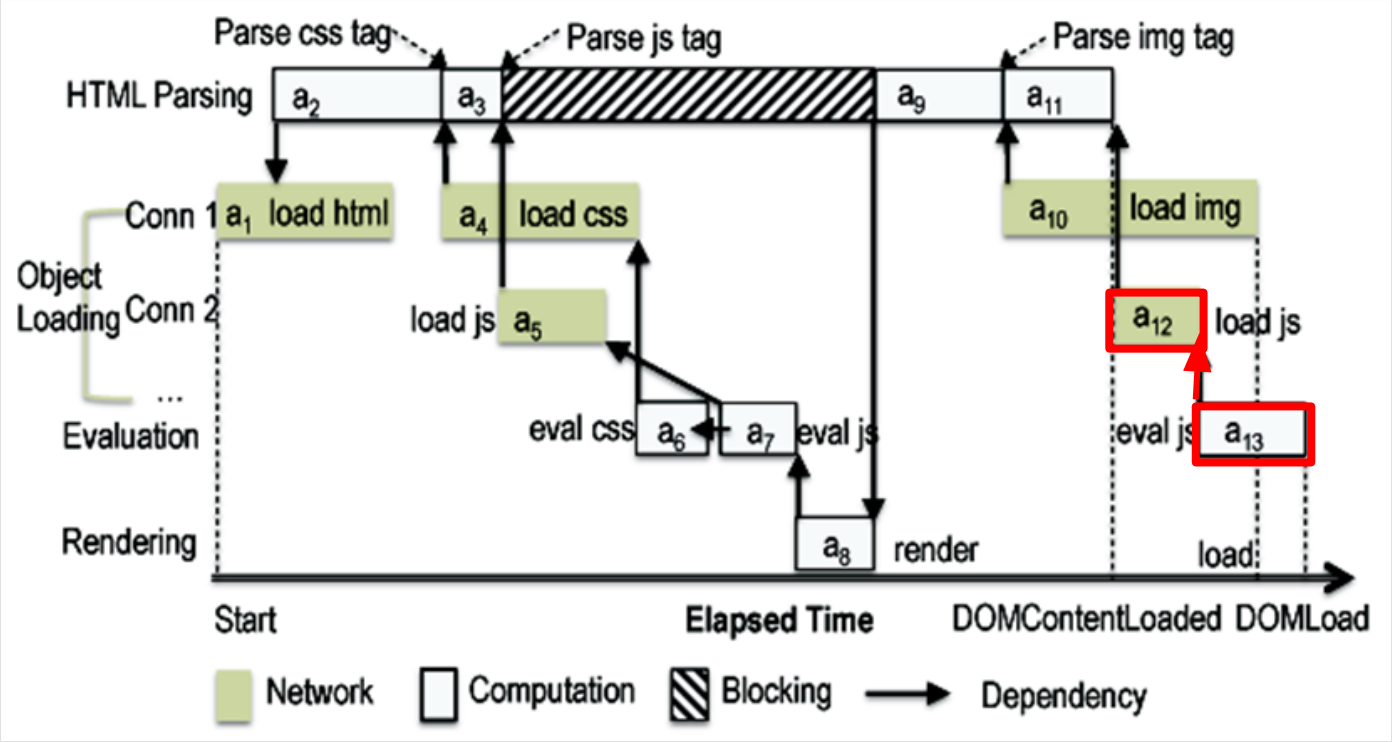
Critical path analysis

Critical path: the longest bottleneck path.



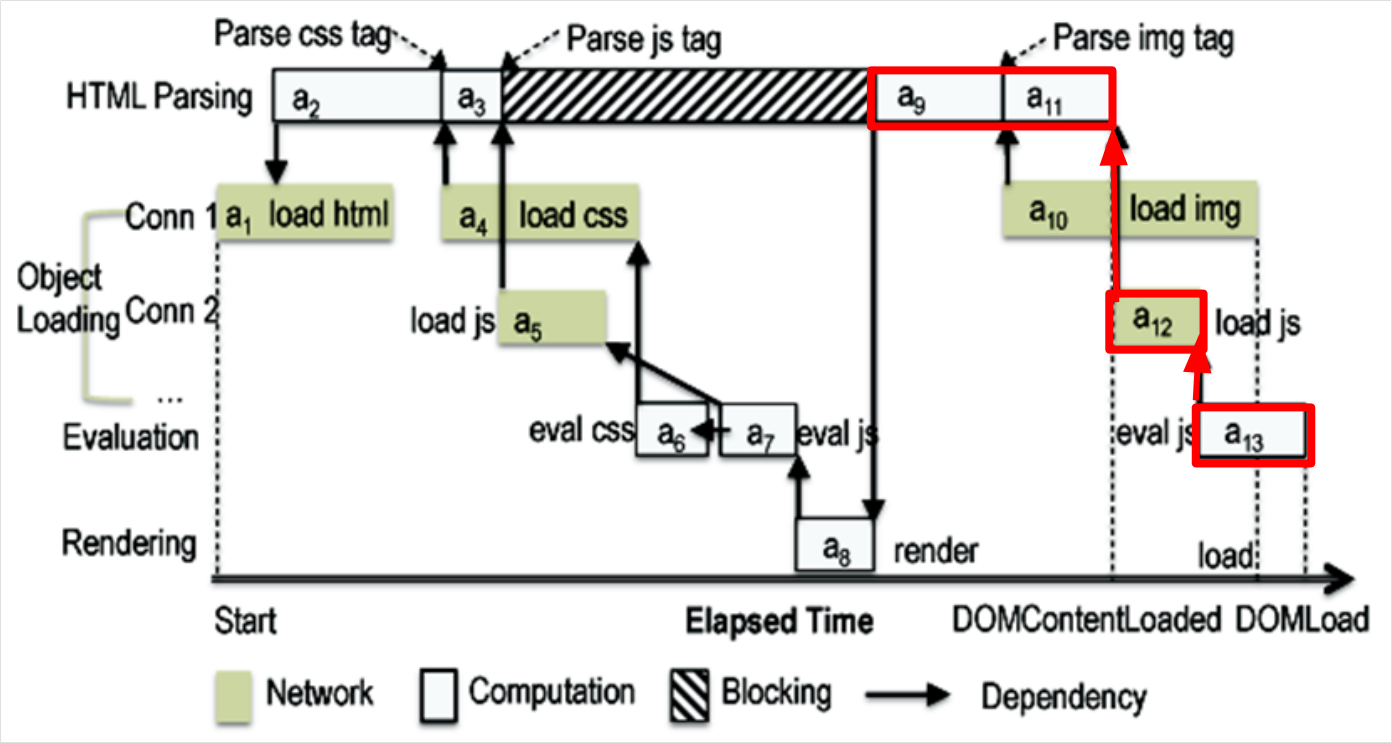
Critical path analysis

Critical path: the longest bottleneck path.



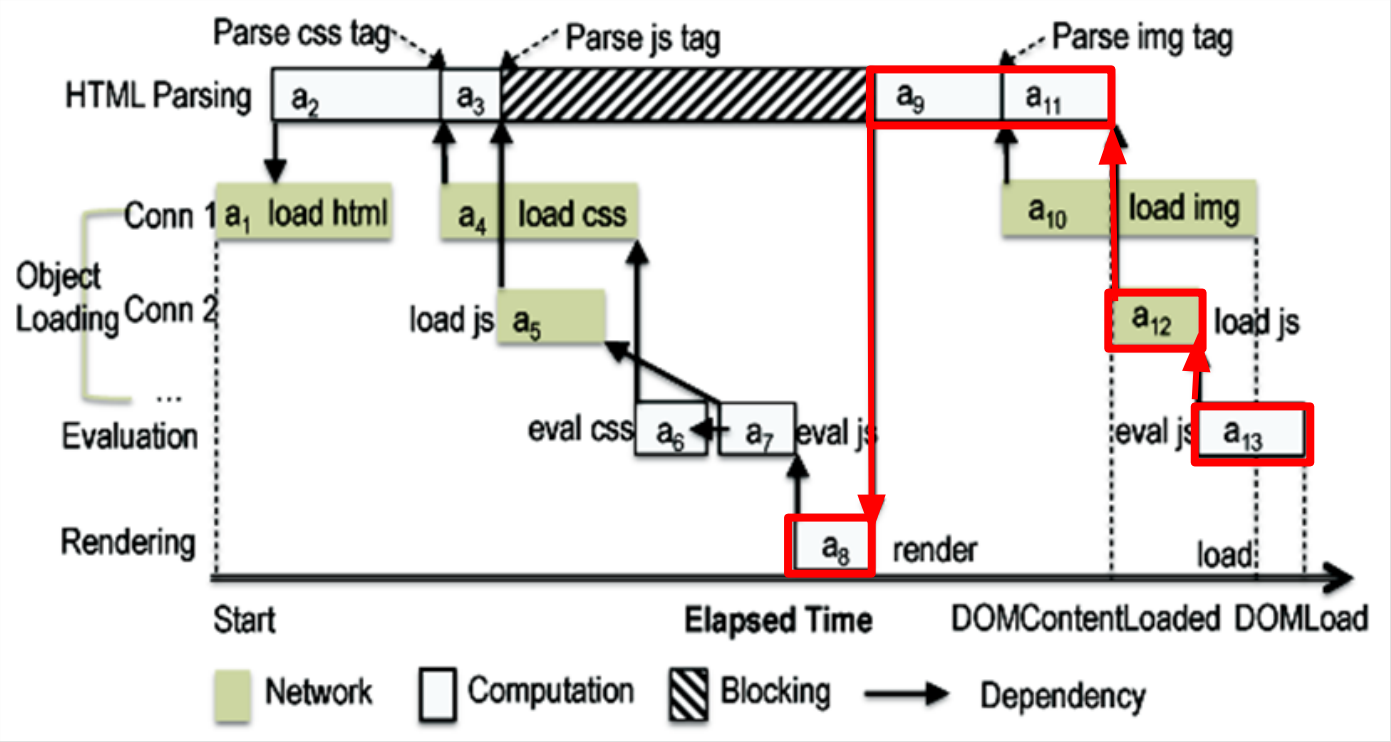
Critical path analysis

Critical path: the longest bottleneck path.



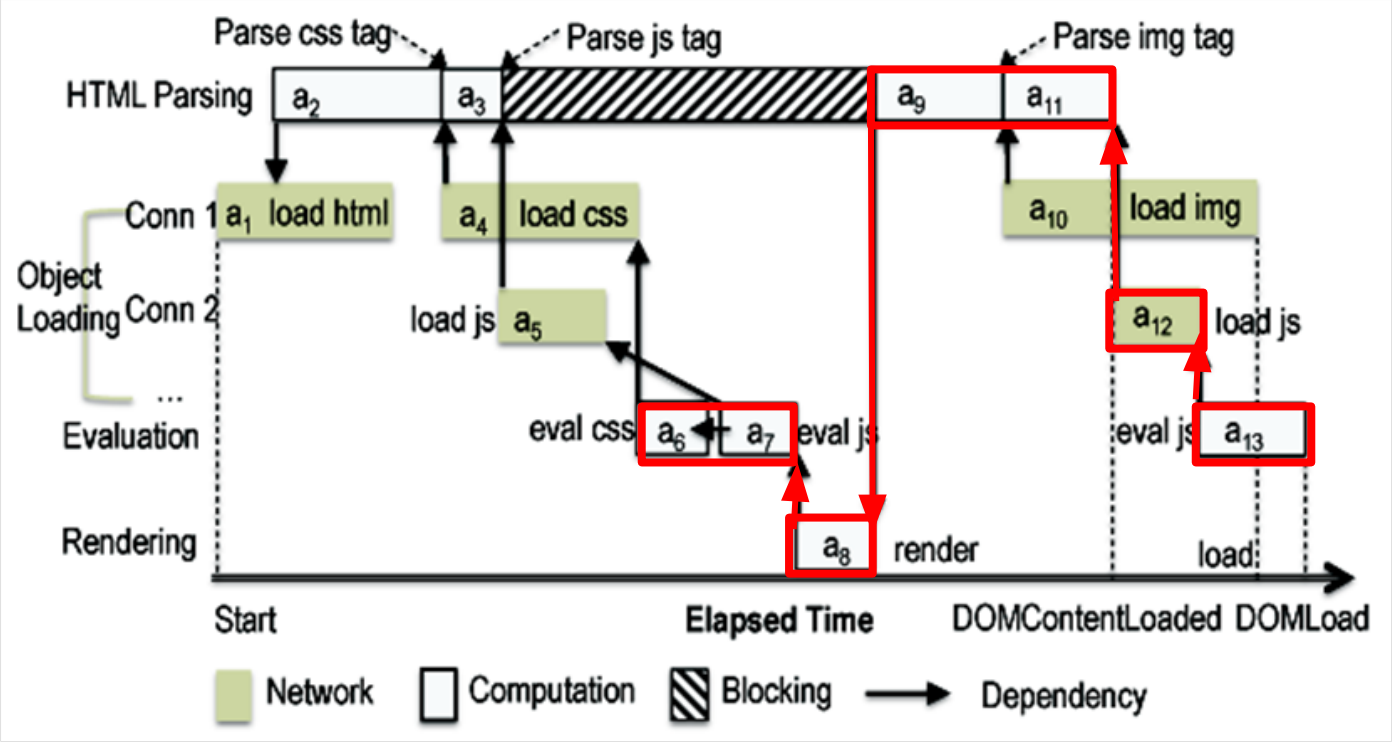
Critical path analysis

Critical path: the longest bottleneck path.



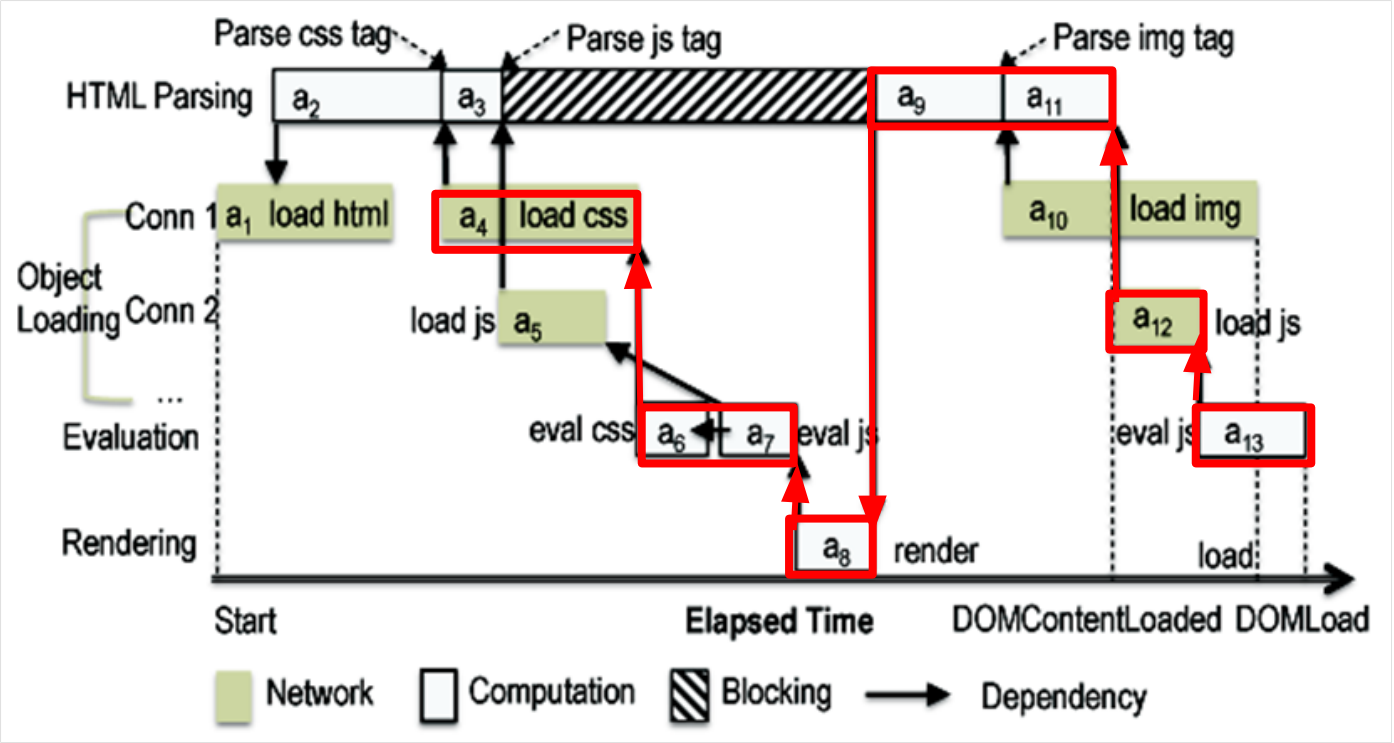
Critical path analysis

Critical path: the longest bottleneck path.



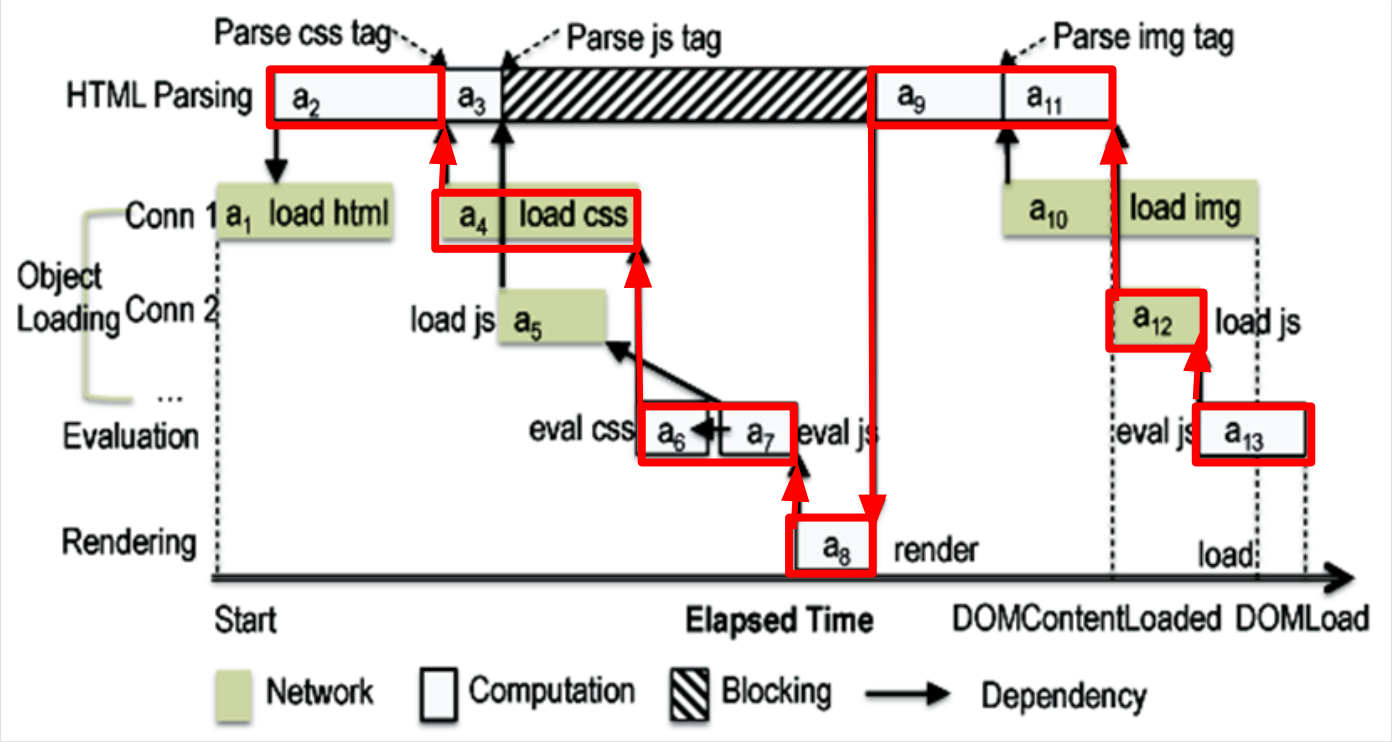
Critical path analysis

Critical path: the longest bottleneck path.



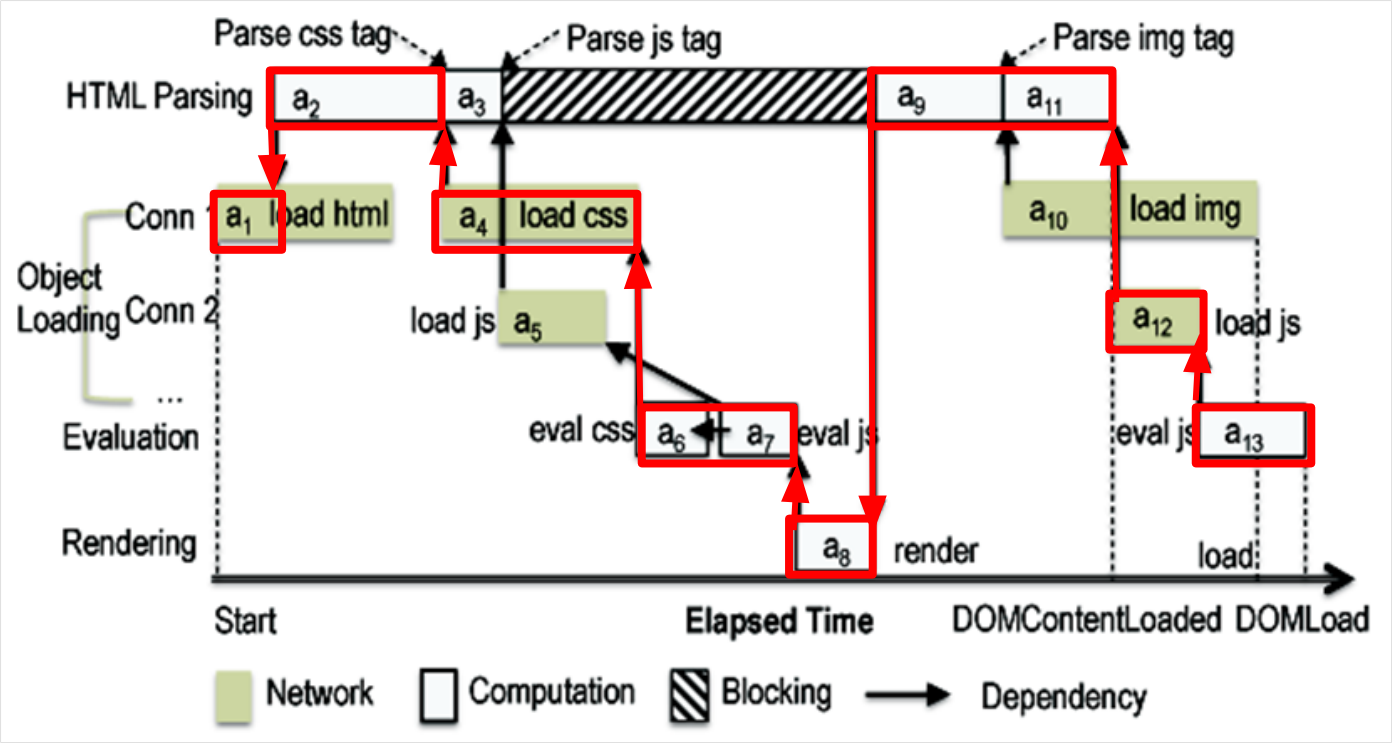
Critical path analysis

Critical path: the longest bottleneck path.



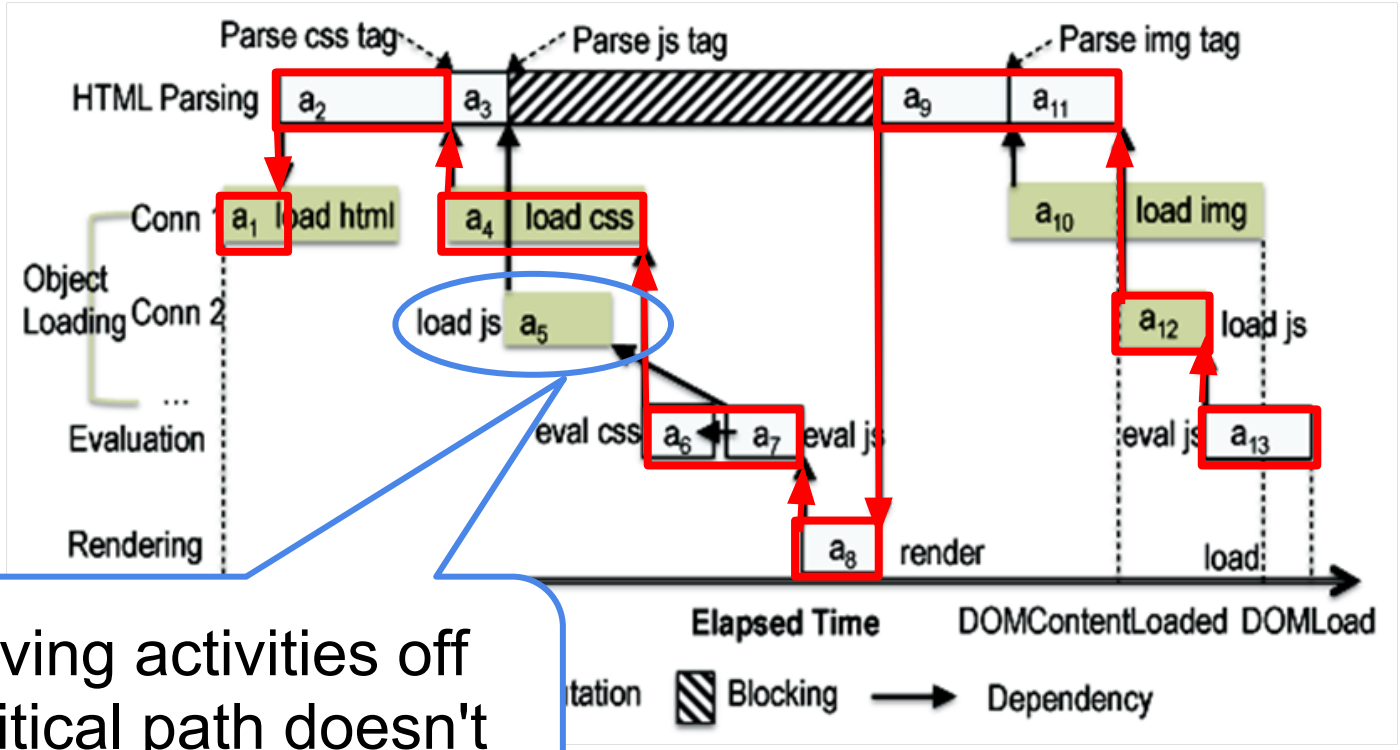
Critical path analysis

Critical path: the longest bottleneck path.



Critical path analysis

Critical path: the longest bottleneck path.



Improving activities off the critical path doesn't help page load.

Overview of our work

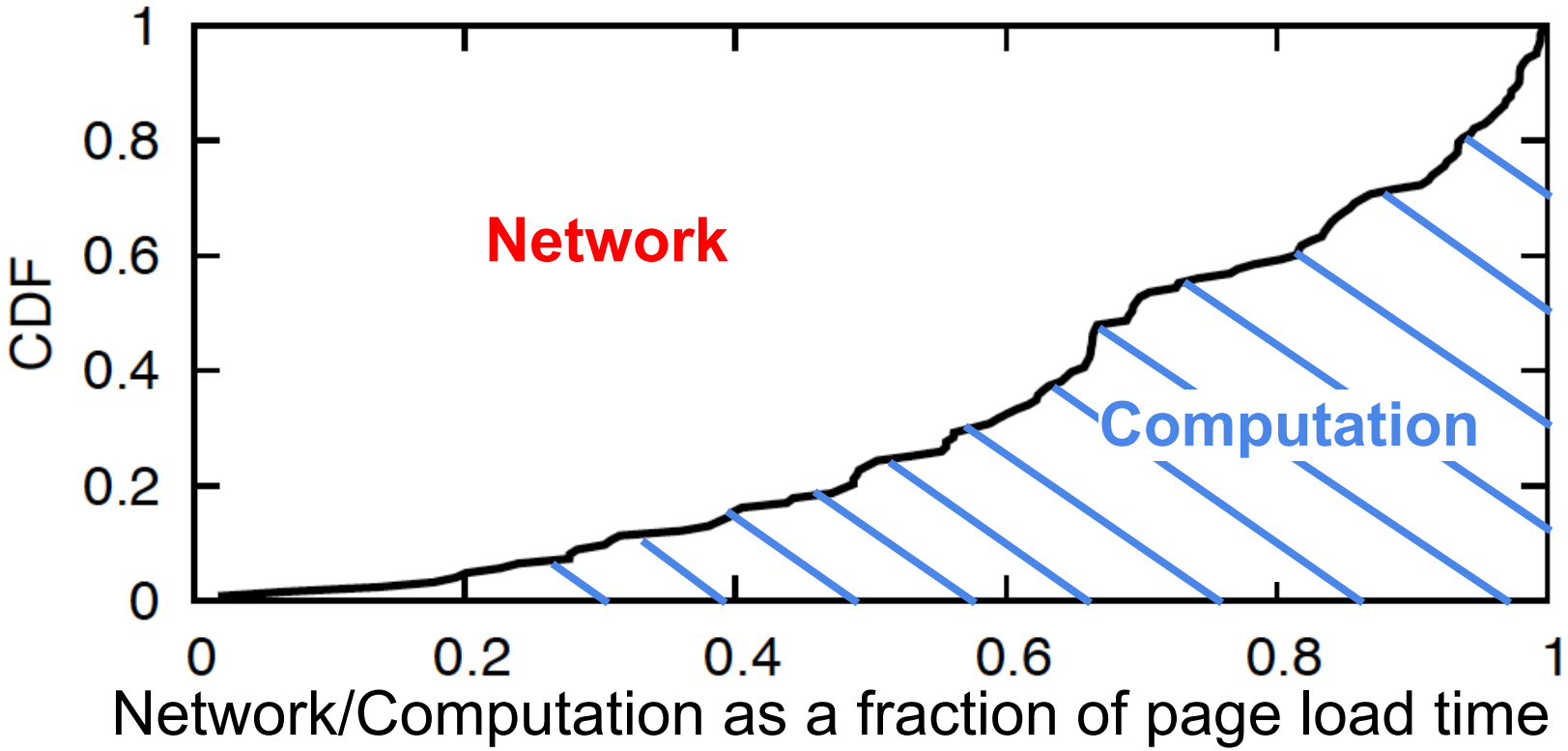
- Model the page load process
- Build the WProf tool
- Study page load with real pages

Experimental setup

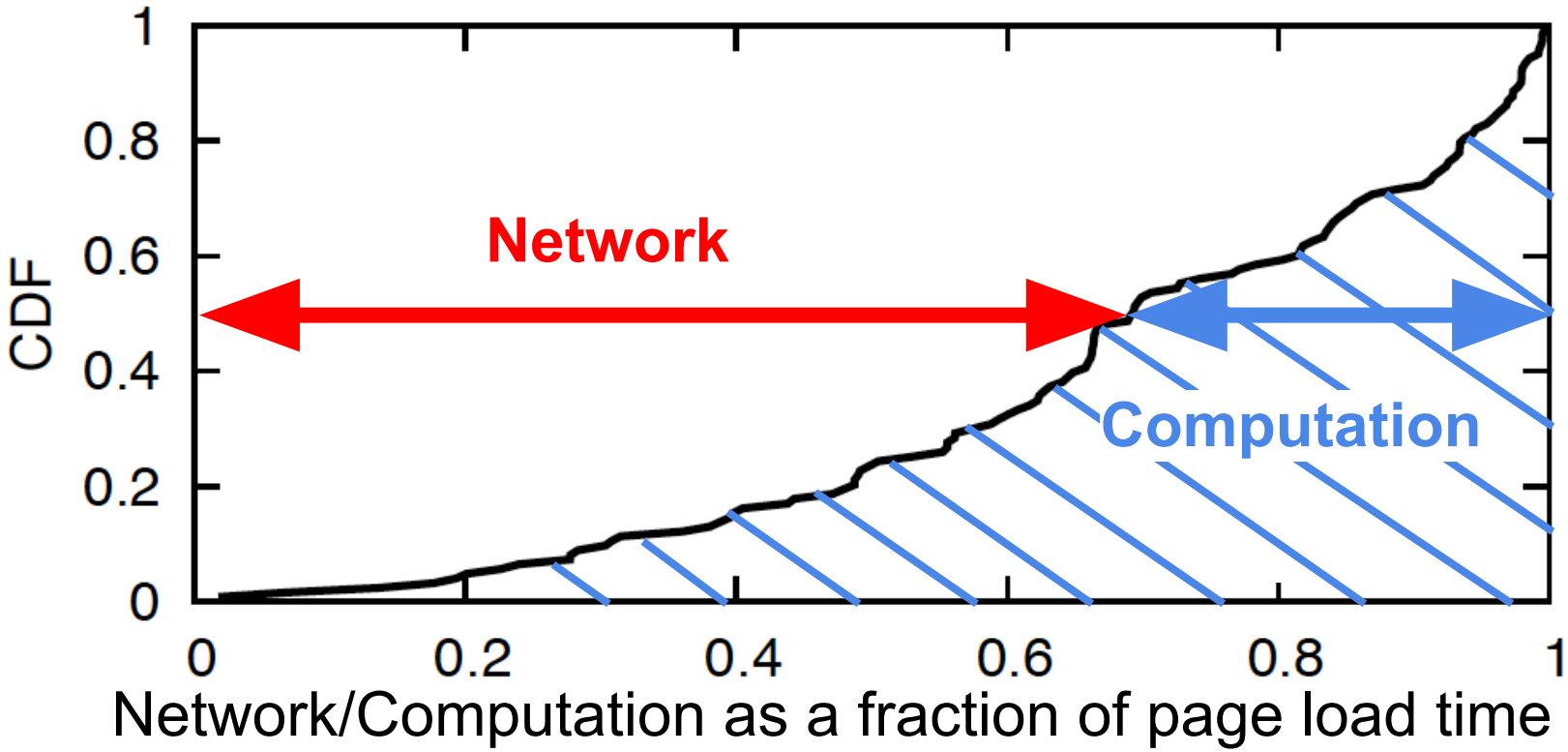
- Location
  - UW Seattle campus network
- Browser
  - WProf-instrumented Chrome
- Web pages
  - 150 out of top 200 Alexa pages
- Page load time
  - Minimum out of 5 repeats

How much does computation contribute to page load time?

Computation is significant



Computation is significant



Computation is ~35% of page load time (median) on the critical path.

How much does caching help page load performance?

How much does caching help?

- Caching eliminates 80% Web object loads
- It doesn't reduce page load time as much

How much does caching help?

- Caching eliminates 80% Web object loads
- It doesn't reduce page load time as much
- Caching only eliminates 40% Web object loads on the critical path

Summary of other results

- Most object downloads are not critical
- JS blocks parsing on 60% top pages
- SPDY doesn't help much as expected
- Minification with mod_pagespeed doesn't reduce received bytes on the critical path

Related work

- Industry tools
  - DevTools, Pagespeed Insights
- Academic
  - WebProphet [NSDI'2010]
    - Only consider network time

Conclusion

- Model page load process
- WProf automatically extracts dependencies and analyzes critical paths
- WProf can be used to
  - Understand performance of any page load
  - Explain behaviors of current optimizations
  - Perform what-if analysis

Project website: wprof.cs.washington.edu