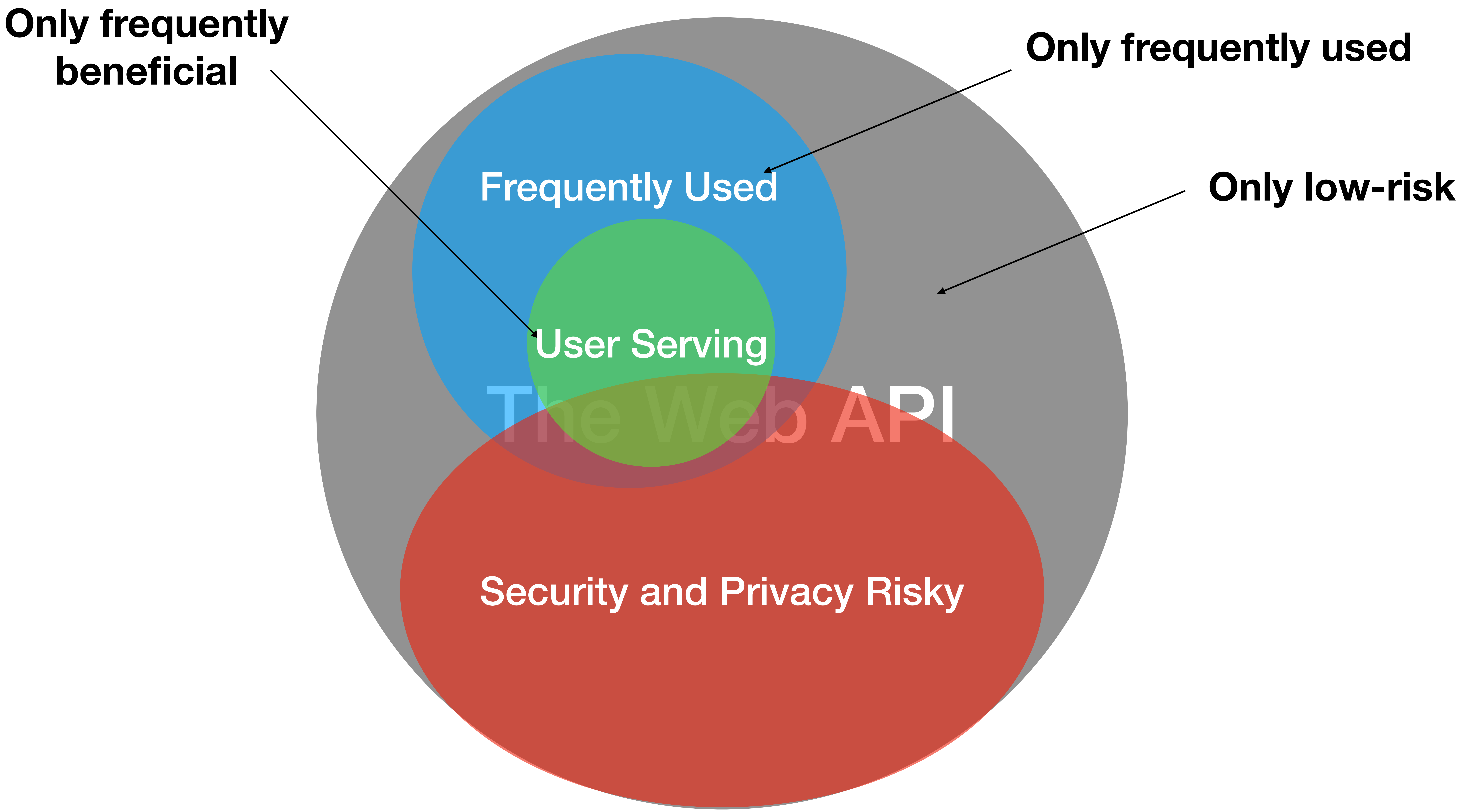
Improving Web Privacy And Security with a Cost-Benefit Analysis of the Web API

Pete Snyder
Thesis Defense

Committee: Christopher Kanich
Venkat Venkatakrishnan
Jakob Eriksson
Stephen Checkoway
Damon McCoy


Only frequently beneficial
Only frequently used
Only low-risk
Frequently Used
User Serving
The Web API
Security and Privacy Risky

Thesis Questions

- Q1: Can we quantitatively distinguish between high and low benefit, and high and low cost, browser features?
- Q2: Can we use this information to improve privacy and security for web users?

Outline

- Background
- Measuring use
- Measuring cost vs. benefit
- Applying findings to “current web”
- Applying findings to “future web”

Outline

- Background
- Measuring use
- Measuring cost vs. benefit
- Applying findings to “current web”
- Applying findings to “future web”

What is the Web API?

- Browser implemented functionality
- Provided to websites as JavaScript methods, events, structures
- Sites authors use these browser capabilities to create interactive sites
- Cross browser (mostly)

What the Web API Is Not

- Internals (networking stack, TLS, etc.)
- Browser interface
- Extensions
- Plugins
- Static documents
- (generally) anything browser specific

What is In the Web API?

- Document manipulation
- AJAX / server requests
- Cookies
- Browser navigation

- Complex graphics animations
- WebGL
- Cryptographic operations
- Parallel operations
- Font operations
- Styling / presentation

- Ambient light sensing
- Peer-to-peer networking
- Audio synthesis
- “Beacons”
- Geolocation
- Gamepads
- Vibration
- High resolution timers
- DRM
- SVG animations
- Speech synthesis
- Battery status

- Virtual reality support
- Selection events
- Fetch API
- Shared memory
- ResourceStats API
- Gesture support
- Pause Frame API
- CSS Paint API
- WebUSB
- Device Memory
- Server Timing
- etc.

Who Defines Web Standards?

- De Jure Standardzation
  - W3C, WHATWG, Khronos Group
  - E.g. WebGL, Web USB, WebVR
- De Facto Standardization
  - Browser competition, retroactive de facto standardization
  - E.g. innerHTML, WebExtension, early DOM standards

Terms

- **Feature**
  A single, JS accessible, function, data structure or event
- **Standard**
  A set of “features”, defined in a standards document (or subsection of a standards document), designed to accomplish a similar set of goals
- **Web API**
  Set of every feature in every standard, or union of all “standards”

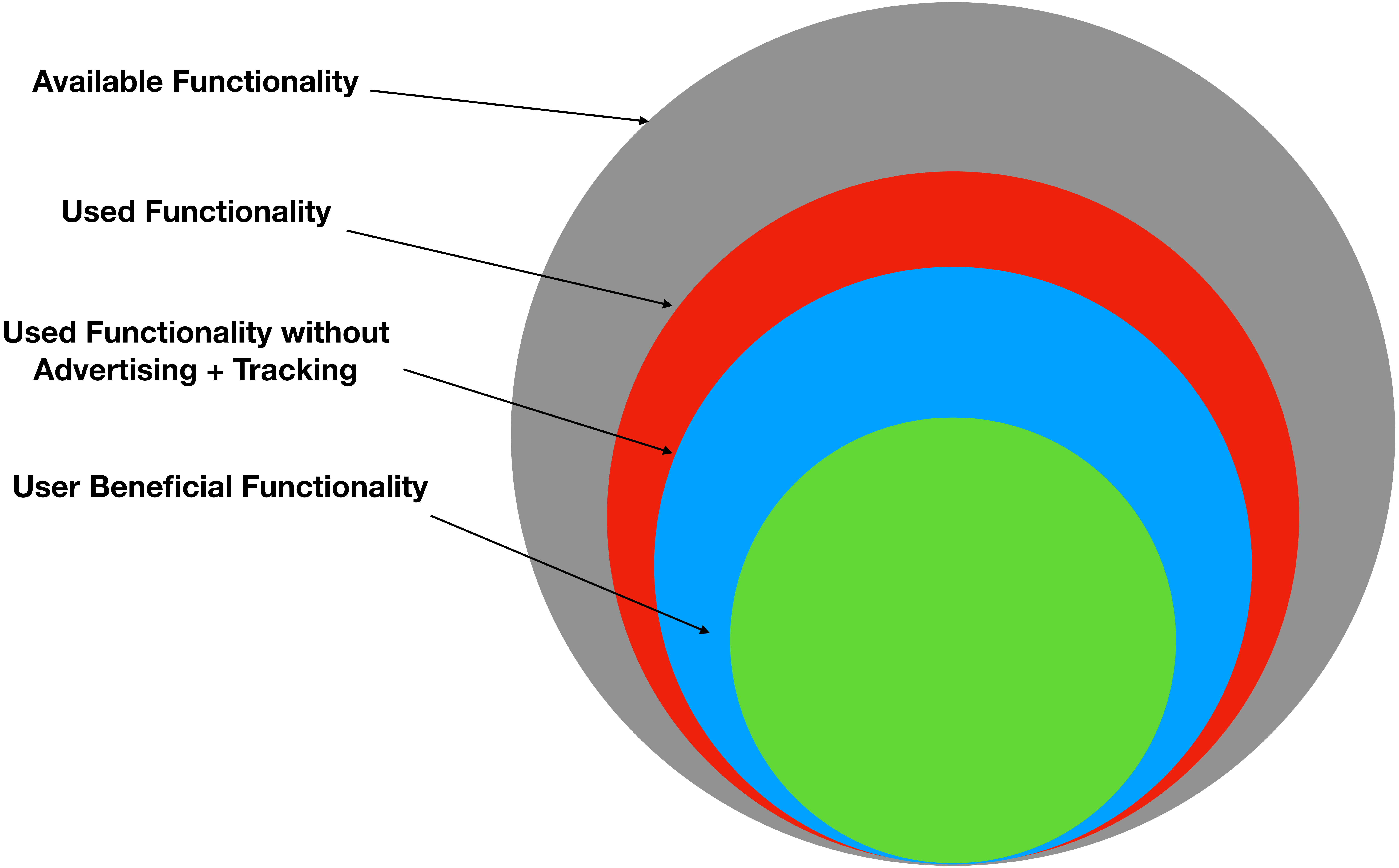
Outline

- Background
- <u>Measuring use</u>
- Measuring cost vs. benefit
- Applying findings to “current web”
- Applying findings to “future web”

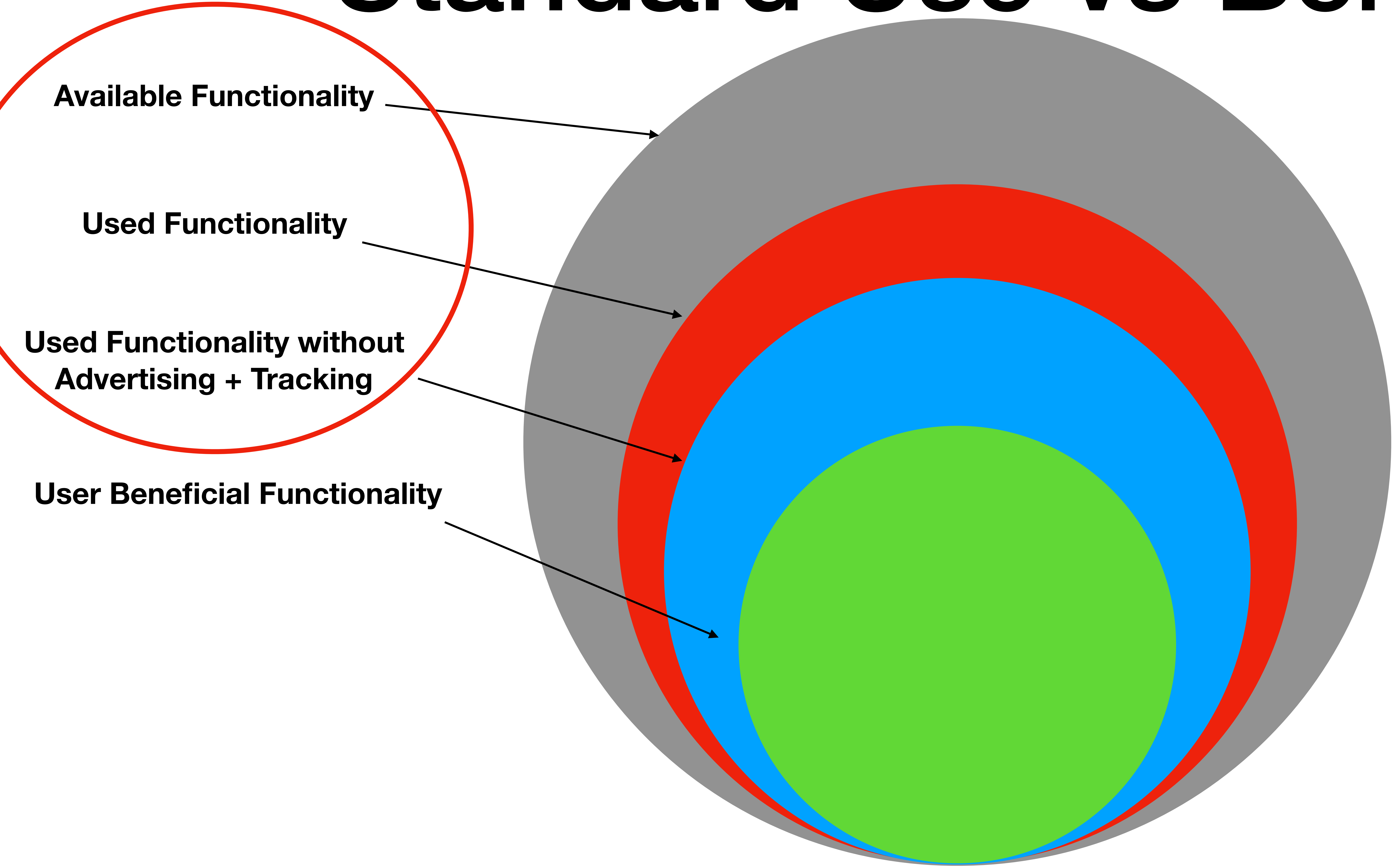
Measuring Feature Use

Snyder, Peter, Lara Ansari, Cynthia Taylor, and Chris Kanich. "Browser feature usage on the modern web." *IMC 2016*

Standard Use vs Benefit



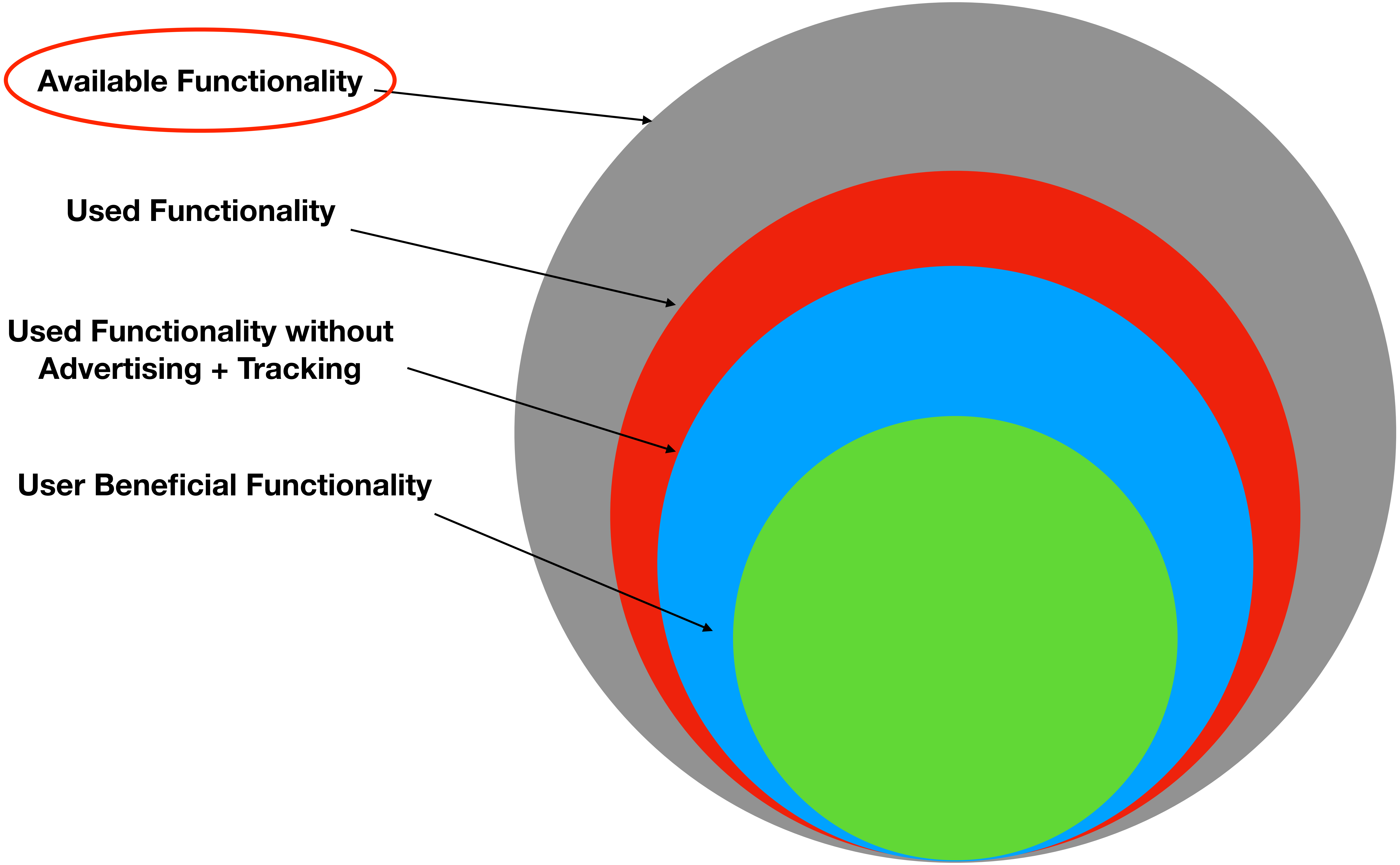
Standard Use vs Benefit



Measuring Feature Use

Methodology

Standard Use vs Benefit

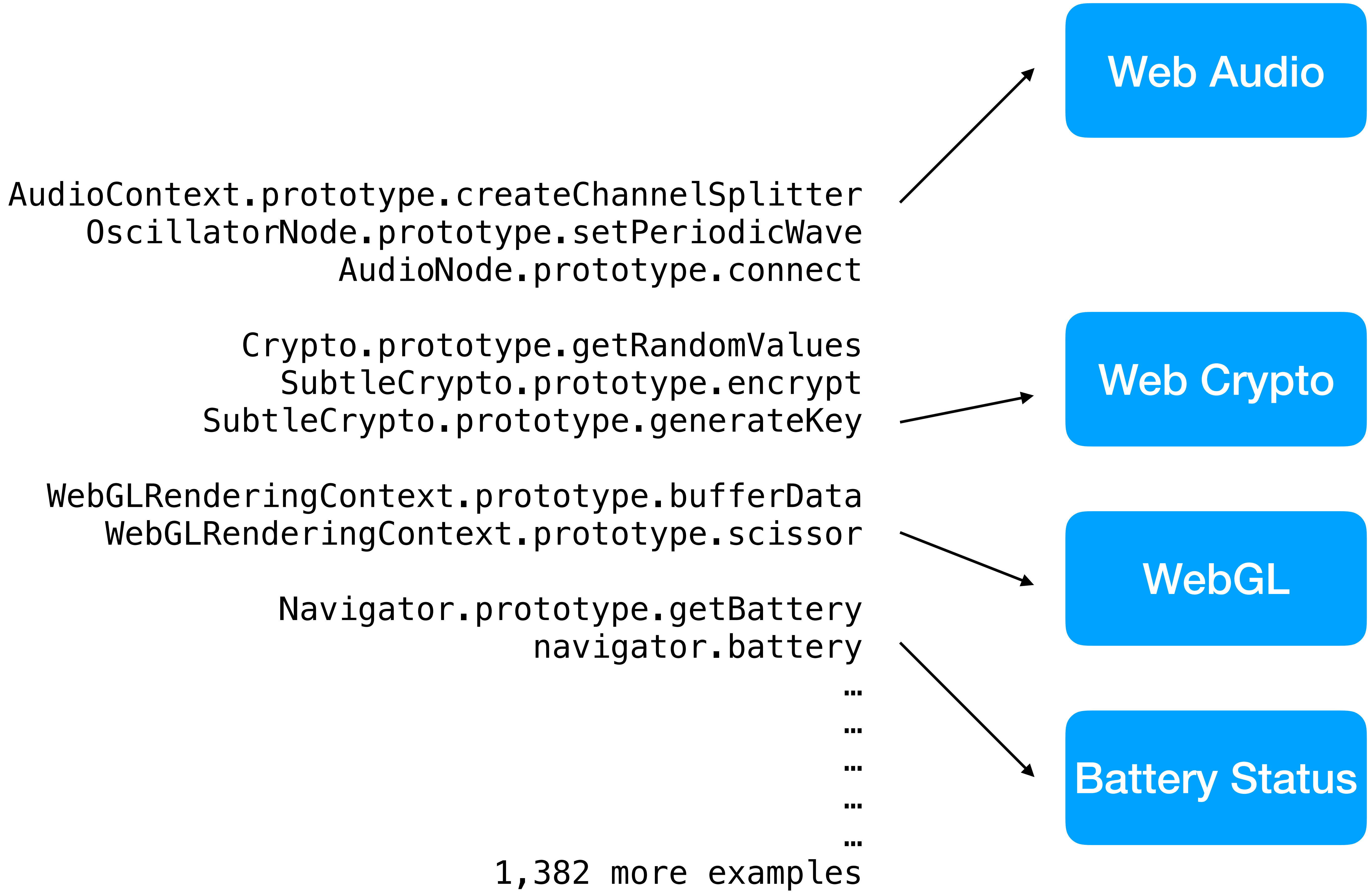


Available Functionality: Data Set

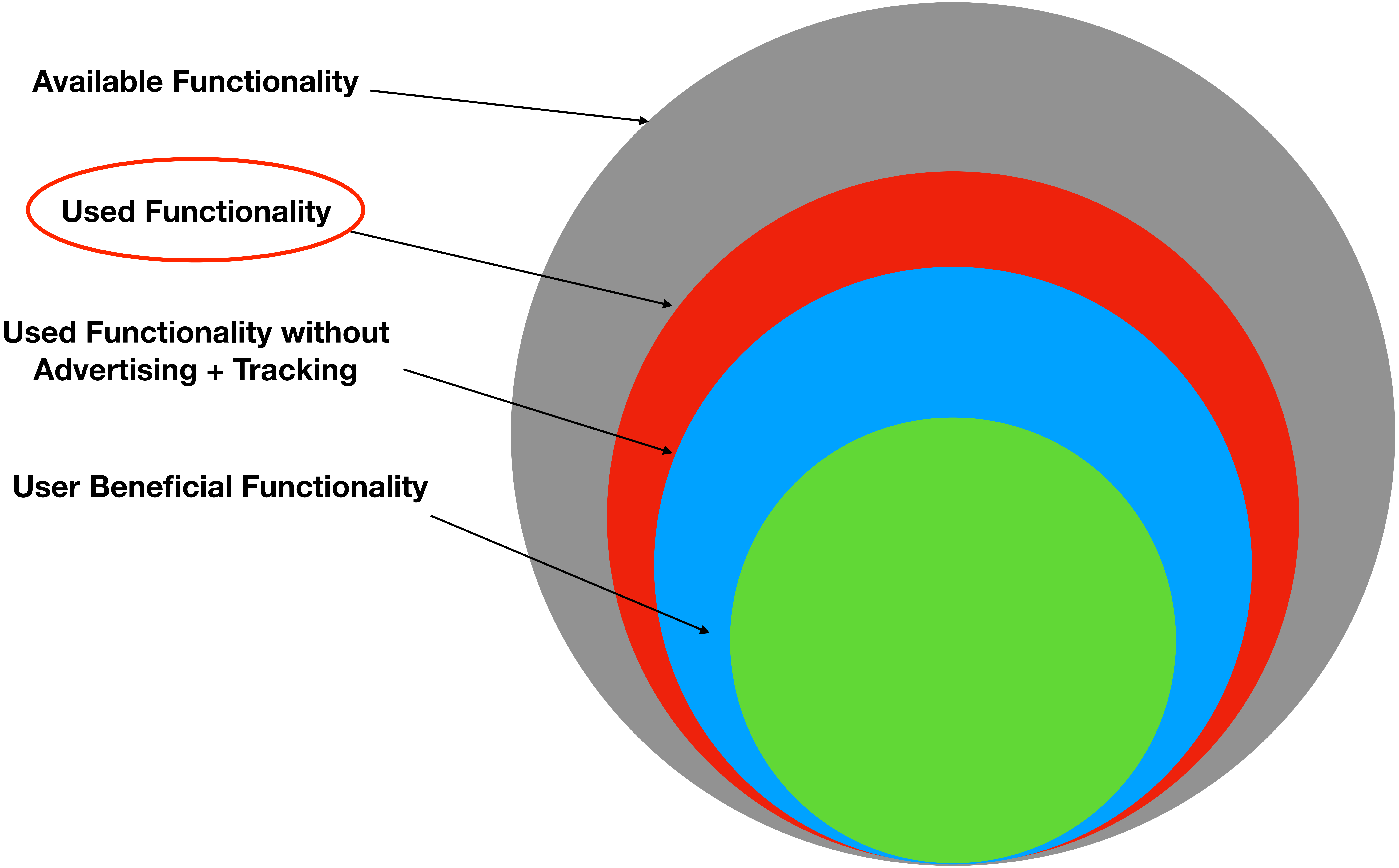
- Representative Browser
  - Firefox 43.0.1
  - Open source
  - Standards focused

Available Functionality: Data Set

- Firefox WebIDL
- 1,392 **features**
- 74 **standards** and sub-standards



Standard Use vs Benefit



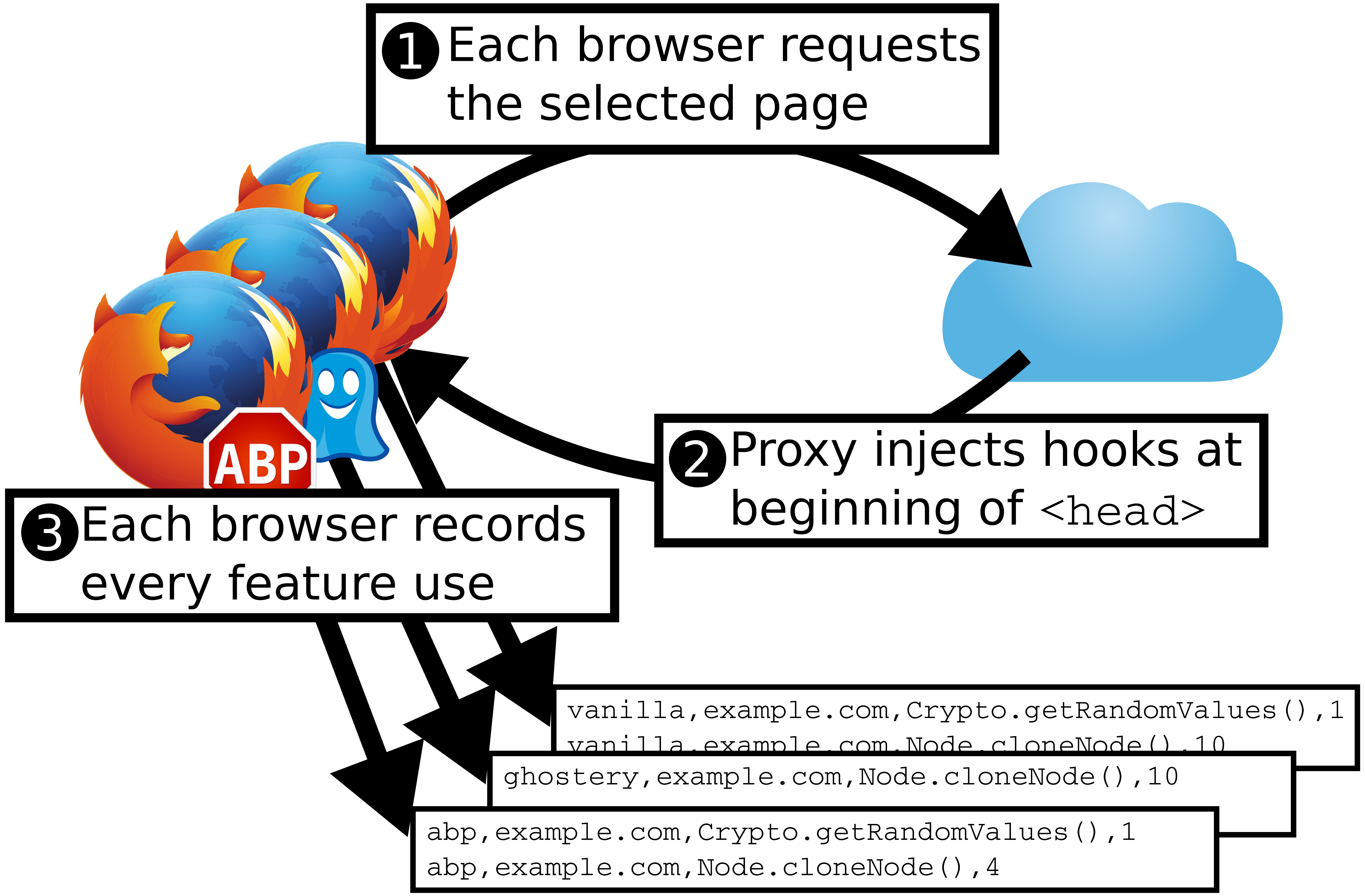
Problem Area Bounds

- Anonymous and {no, low} trust environments
- Only consider costs and benefits to users
- Site and developer interests left out

Used Functionality: Methodology

- Alexa 10k to represent the web
- Too much for a manual review
- Javascript makes static analysis difficult
- Automation with extension-based measurement

Measuring Code Injection

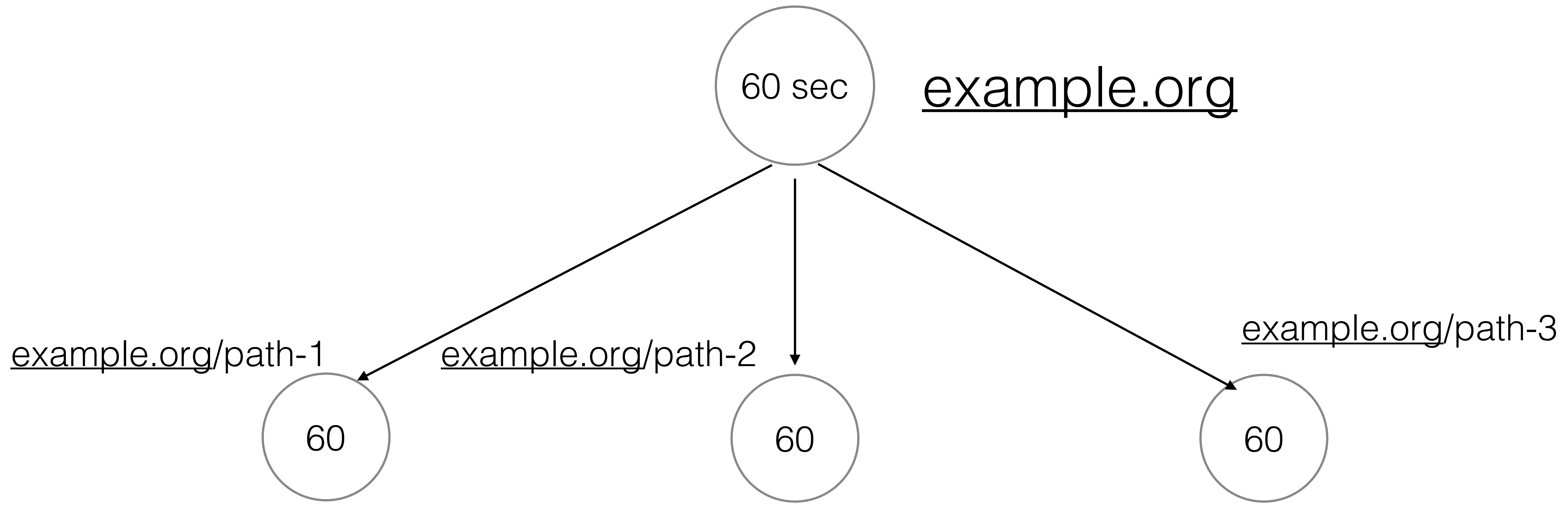


Measuring Code Injection

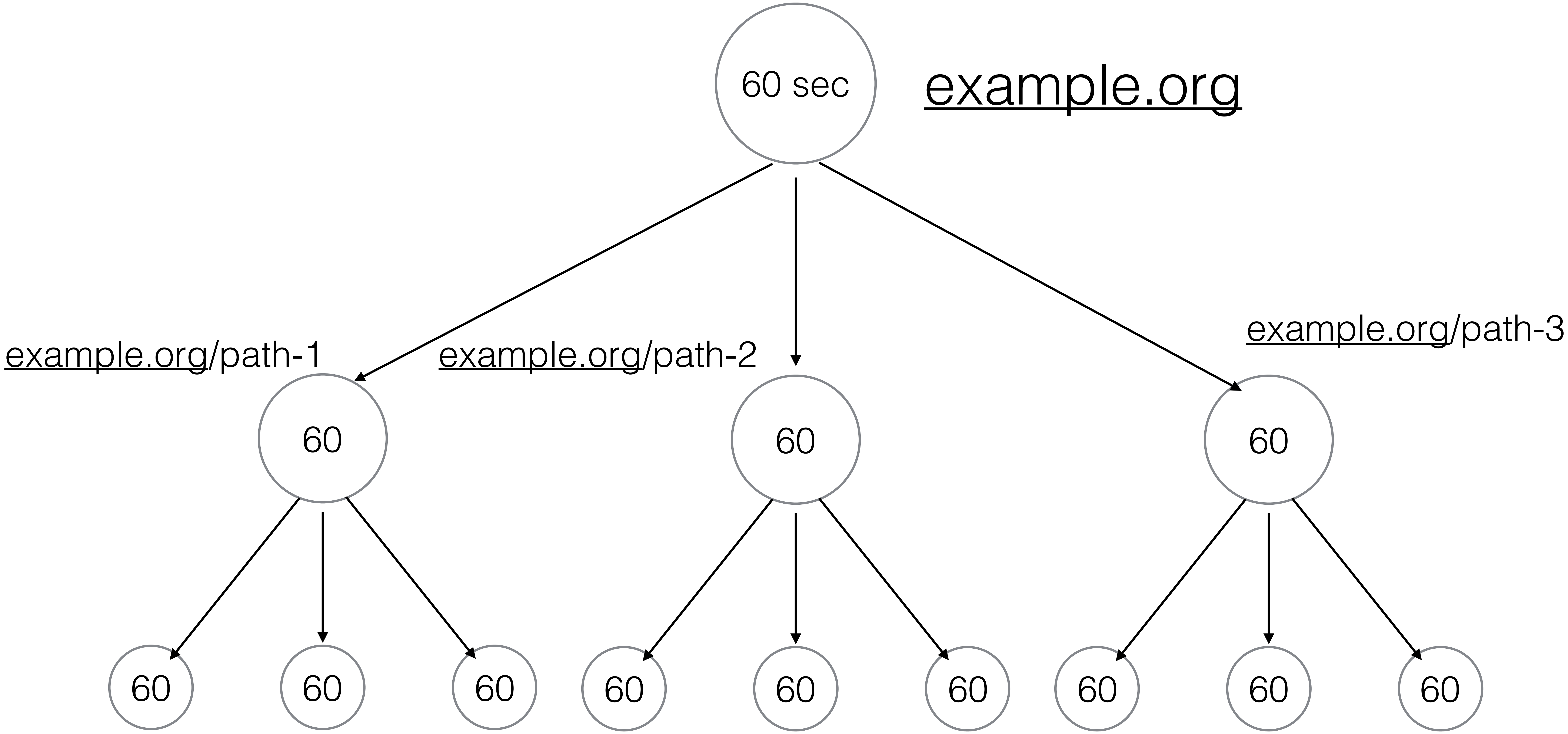
60 sec

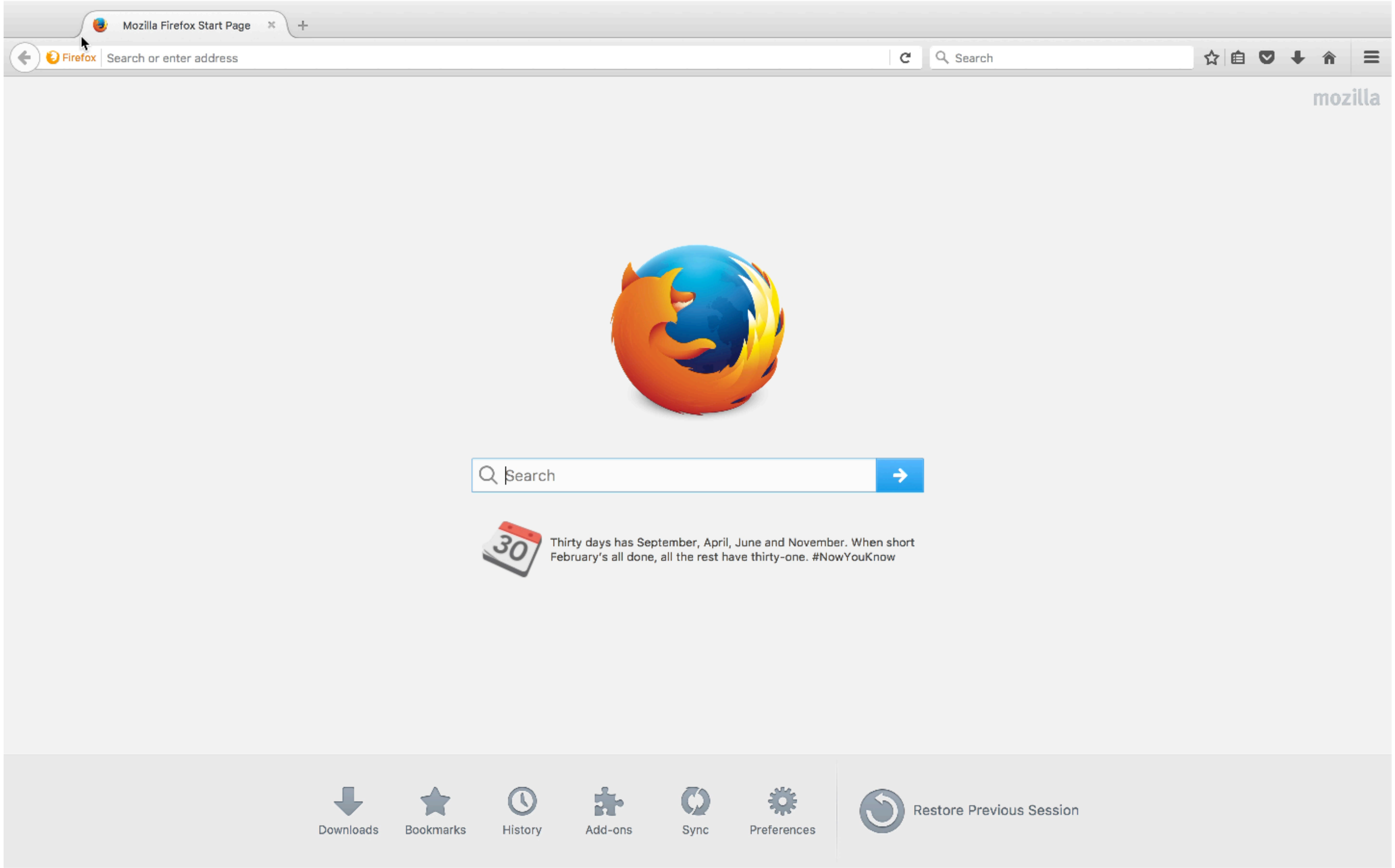
example.org

Measuring Code Injection



Measuring Code Injection



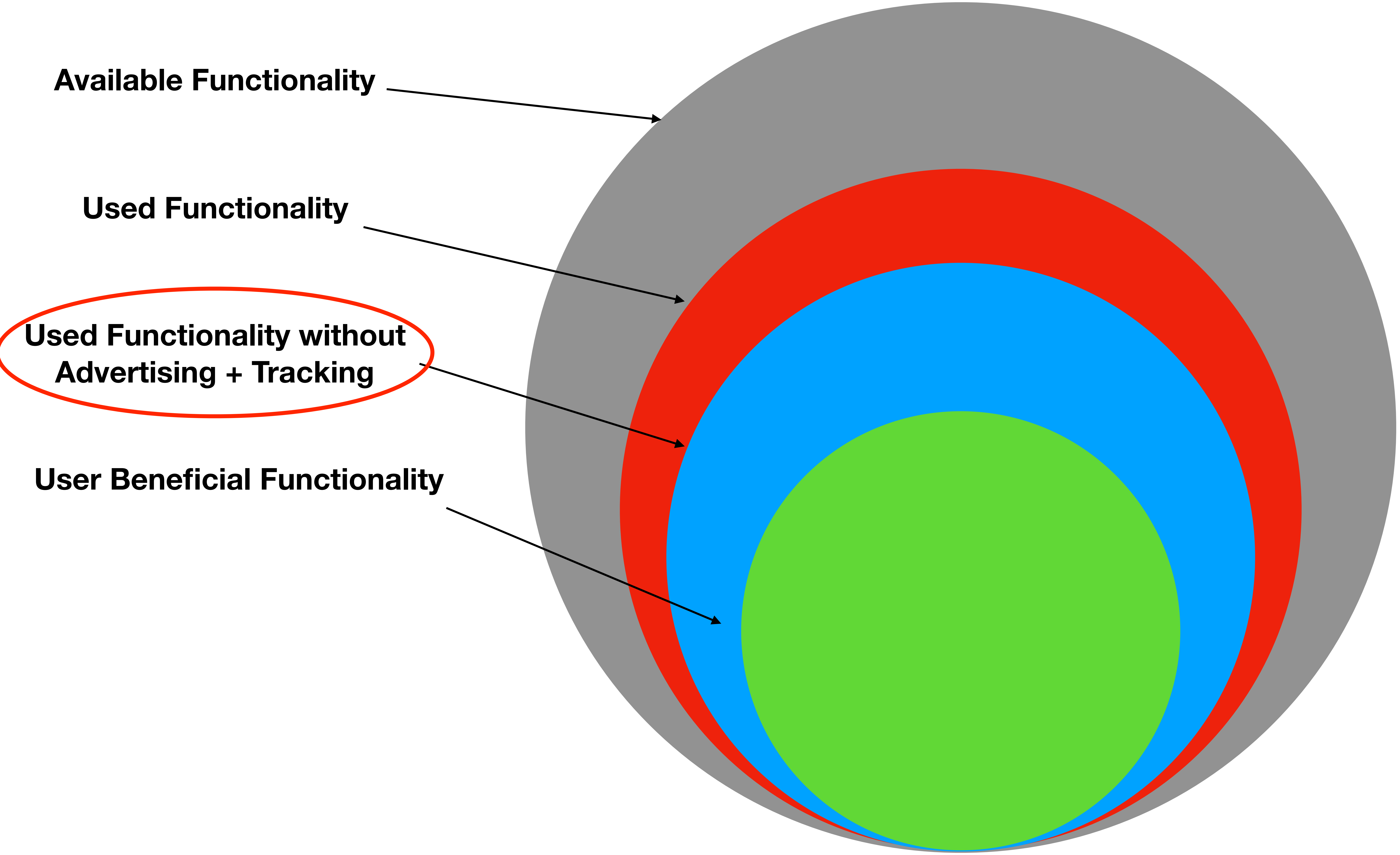

Mozilla Firefox Start Page
Firefox
Search or enter address
Search
mozilla
Search
Thirty days has September, April, June and November. When short February's all done, all the rest have thirty-one. #NowYouKnow
Downloads
Bookmarks
History
Add-ons
Sync
Preferences
Restore Previous Session

Automated Measurement

- 5 times per domain
- Every site in Alexa 10k
- 4 browser configurations

Domains measured	9,733
Total website interaction time	480 days
Web pages visited	2,240,484
Feature invocations recorded	21,511,926,733

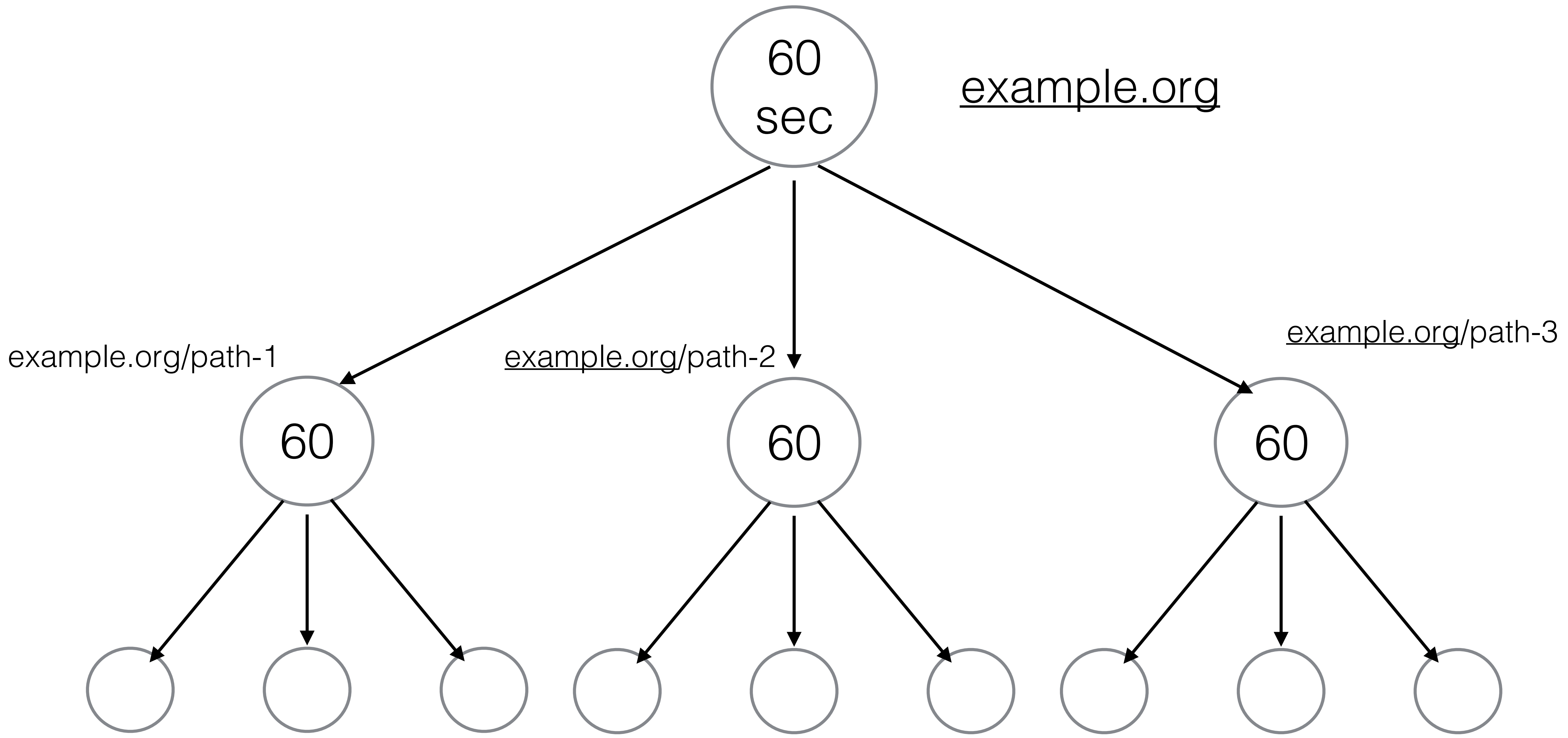
Standard Use vs Benefit



Removing Tracking and Advertising

Measuring Code Injection

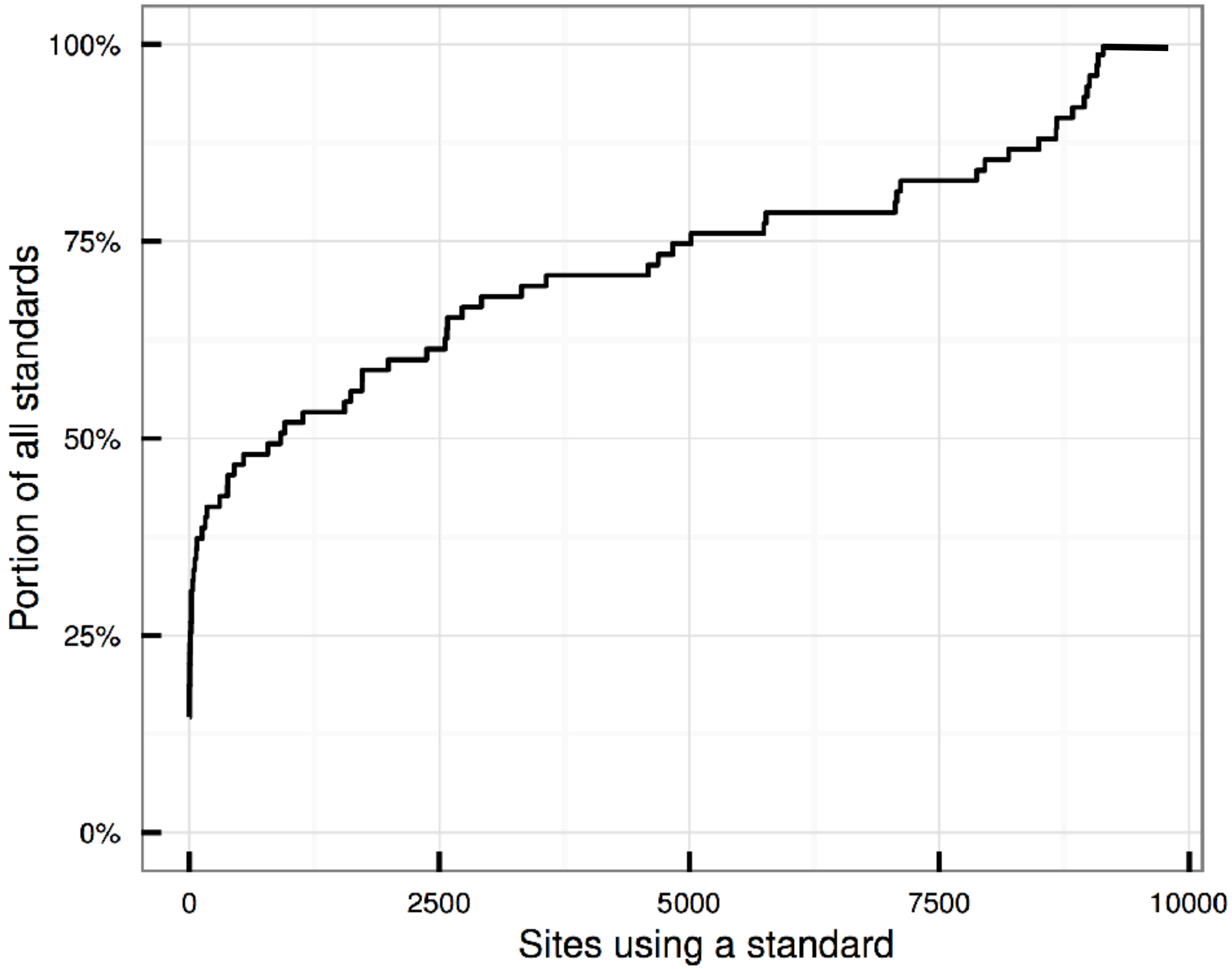
+



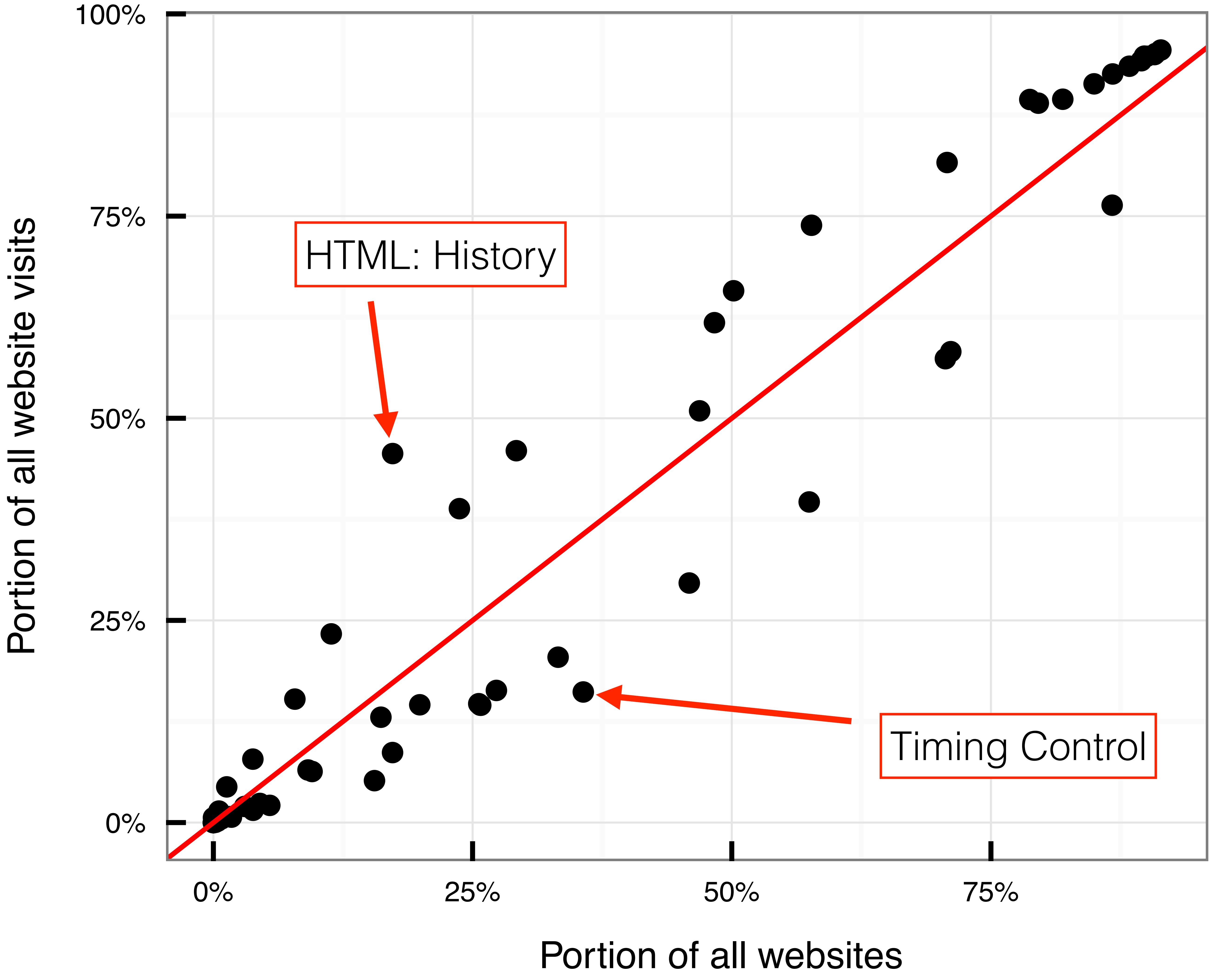
Measuring Feature Use

Results

Standard Popularity

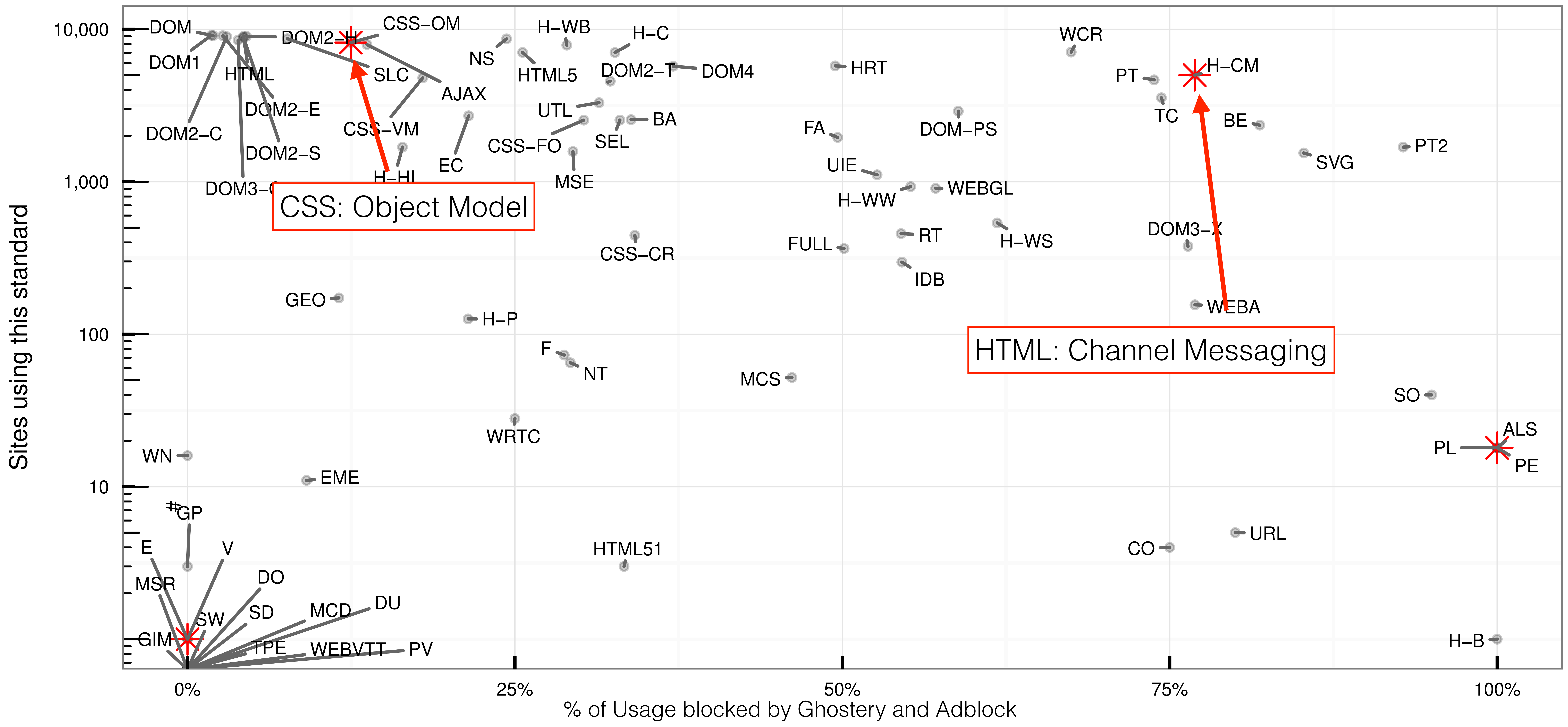


Standard vs Site Popularity

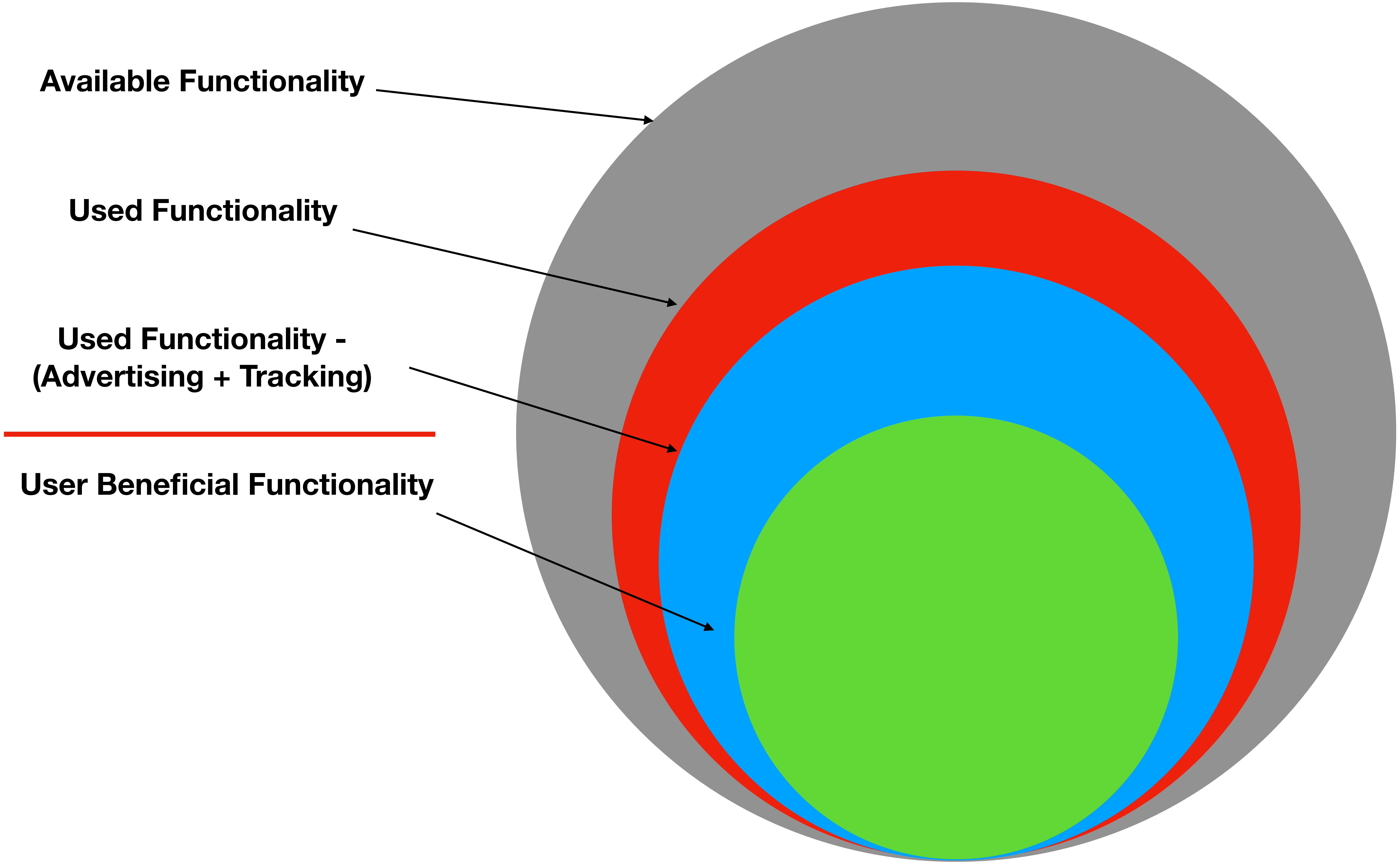


Standard Popularity by Date

Standard Popularity vs Blocking



Standard Use vs Benefit



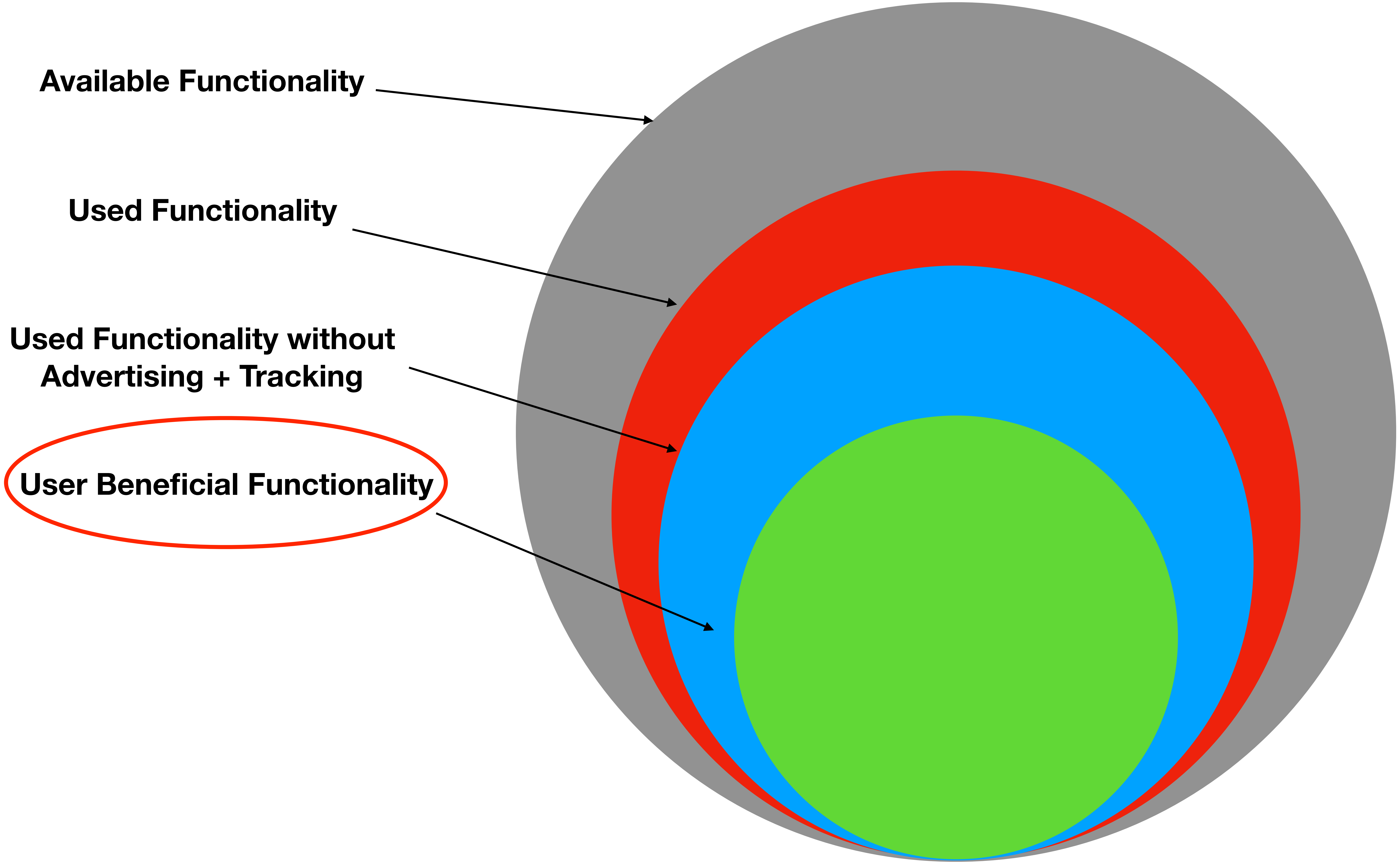
Outline

- Background
- Measuring use
- <u>Measuring cost vs. benefit</u>
- Applying findings to “current web”
- Applying findings to “future web”

Measuring Feature Cost vs. Benefit

Snyder, Peter, Cynthia Taylor, and Chris Kanich. "Most Websites Don't Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security." *CCS, 2017*

Standard Use vs Benefit



Feature Cost vs. Benefit

Methodology

Feature Cost vs. Benefit: Methodology

- Measuring Benefit
  - Number of sites that “need” a feature
- Measuring Cost
  - References in peer-reviewed literature
  - Recent related vulnerabilities
  - Additional code complexity

Measuring Feature Benefit (1/3)

- **Intuition**: Web API standards that are less frequently needed to accomplish user-serving tasks are less beneficial to users.
- **Metric**: What % of websites break when a standard is removed from the browser?
  - ↑ means more beneficial, ↓ means less beneficial
- Only considers benefit to browser users (not site owners)
- Only considering the anonymous / no-trust case

Measuring Feature Benefit (2/3)

- For each of the 74 standards in the browser:
  - Randomly select 40 sites using the standard
  - Have two students independently visit the site for 60 seconds
  - Remove the standard from the browser, revisit site for 60 seconds
  - Record if they were able to accomplish "the site's main purpose"
  - 96.74% agreement between testers

Measuring Feature Benefit (3/3)

- Ranking System
    - 1: No visible difference
    - 2: Some difference, but didn't affect core functionality
    - 3: Core functionality affected
- 96.74% agreement between testers (1 & 2 vs 3)

Feature Removal Strategy

- **Problem**
  - Removing functions from the environment will break unrelated code
  - Lead to over count in site need
- **Goal**
  - Want to block page access to functionality
  - Have other code run as normal

```
const canvas = document.createElement("canvas");

const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat(
  gl.VERTEX_SHADER,
  gl.MEDIUM_FLOAT
);
console.log(format.precision); // Finger printing

document.getElementById("some-element");
```

```
WebGLRenderingContext.prototype.getShaderPrecisionFormat = null;
const canvas = document.createElement("canvas");

const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat( // Throws
  gl.VERTEX_SHADER,
  gl.MEDIUM_FLOAT
);
console.log(format.precision); // Finger printing

// Never Called
document.getElementById("some-element");
```

```
WebGLRenderingContext.prototype.getShaderPrecisionFormat = () => null;
const canvas = document.createElement("canvas");

const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat(
  gl.VERTEX_SHADER,
  gl.MEDIUM_FLOAT
);
console.log(format.precision); // Throws

// Never Called
document.getElementById("some-element");
```

```
WebGLRenderingContext.prototype.getShaderPrecisionFormat = new Proxy(…);
const canvas = document.createElement("canvas");

const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat( // Proxied "call" operation
  gl.VERTEX_SHADER,
  gl.MEDIUM_FLOAT
);
console.log(format.precision); // Proxied "get" operation

// Code execution continues as expected
document.getElementById("some-element");
```

```
const blockingProxy = new Proxy(function () {}, {
    get: function (ignore, property) {

        if (property === Symbol.toPrimitive) {
            return toPrimitiveFunc;
        }

        if (property === "valueOf") {
            return toPrimitiveFunc;
        }

        return blockingProxy;
    },
    set: function () {
        return blockingProxy;
    },
    apply: function () {
        return blockingProxy;
    },
    ownKeys: function () {
        return unconfigurablePropNames;
    },
    has: function (ignore, property) {
        return (unconfigurablePropNames.indexOf(property) > -1);
    }
});
```

```
WebGLRenderingContext.prototype.getShaderPrecisionFormat = new Proxy(…);
const canvas = document.createElement("canvas");

const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat( // Proxied "call" operation
  gl.VERTEX_SHADER,
  gl.MEDIUM_FLOAT
);

format.get("these").things[3].thatDo().not.exist;

// Code still continues as expected
document.getElementById("some-element");
```

Measuring Benefit Summary

- Only a subset of the standards in Web API is used
- Users only notice when a subset of those standards are removed
- If users don't noticed when they're not available -> not useful

Per-Standard Cost

- Published attacks using the standard
- Past vulnerabilities associated with the standard
- Code complexity added by the standard

Standard Cost: Related Research

- **Intuition**: Functionality frequently leveraged in attacks in academic publications poses a greater cost to S&P.
- **Metric**: How many papers in top research conferences use a standard in their attack?
- Past 5 years of 10 top security conferences and journals (2010-2015)
- USENIX, S&P, NDSS, CCS, ESORICS, WOOT, ACSAC, Cryptology, etc

Standard Cost: Past Vulnerabilities

- **Intuition**: Functionality that has harmed security and privacy in the past should be treated with greater caution.
- **Metric**: How many CVEs have been filed against a standard's implementation in Firefox
- Look for all CVEs against Firefox since 2010
- Where possible, attribute to a standard
- 1,554 CVEs in general, 175 attributable to a standard
- Distinguish CVEs associated with a standard and other parts of the browser

Standard Cost: Past Vulnerabilities

CVE named Web API Standard	117	66.9%
CVE named JS method unique to a Web API Standard	32	18.3%
CVE named C++ method uniquely tied to a Web API Standard	21	12%
CVE named functionality uniquely associated with Web API Standard	5	2.8%
Total	175	100%

Standard Cost: Code Complexity

- **Intuition**: Functionality that adds greater complexity to the browser code base poses a greater cost to S&P.
- **Metric**: How many lines of code are uniquely in the browser to support each browser standard?
- Static analysis of C++ implementation code in Firefox

Standard Cost: Code Complexity

1. Build call-graph using Clang and Mozilla's DXR tools

2. Identify entry point into call graph for each JS end point in the standard

3. Remove those entry points and identify newly orphaned nodes

4. Attribute LOC in orphaned nodes as being code uniquely attributable to the standard

5. Remove newly orphaned nodes, GOTO 4

Standard Cost: Code Complexity

Standard Cost: Code Complexity

- Caveats and short comings
  - Does not include third party code
  - Does not include code shared between standards
- Metric: # lines of code unique to standard in Web API

Methodology Summary

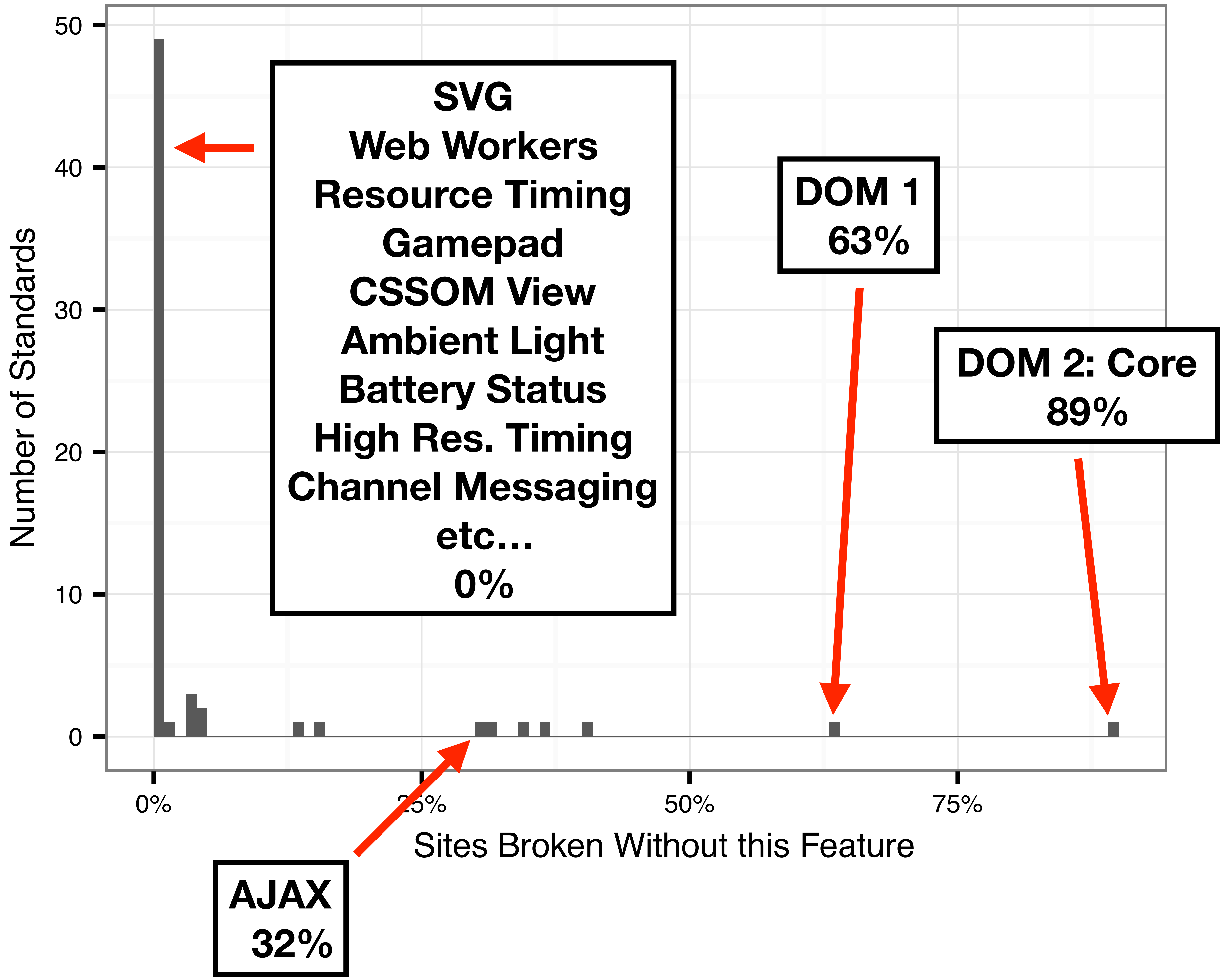
- Alexa 10k as representative of the internet
- Firefox 43.0.1 as representative of browsers
- One metric for measuring benefit
  - Site break rate
- Three metrics for measuring cost
  - CVEs, academic literature, lines of code

Feature Cost vs. Benefit

Results

Standard Benefit

- Most standards provide very little benefit to browser users
- For **60%** of standards, no measurable impact on browsing when they're removed
- Sometimes because the standard was never used (e.g. WebVTT)
- Sometimes because the standard is intended to not be visible (e.g. Beacon)



Standard Cost: Related Research (1/2)

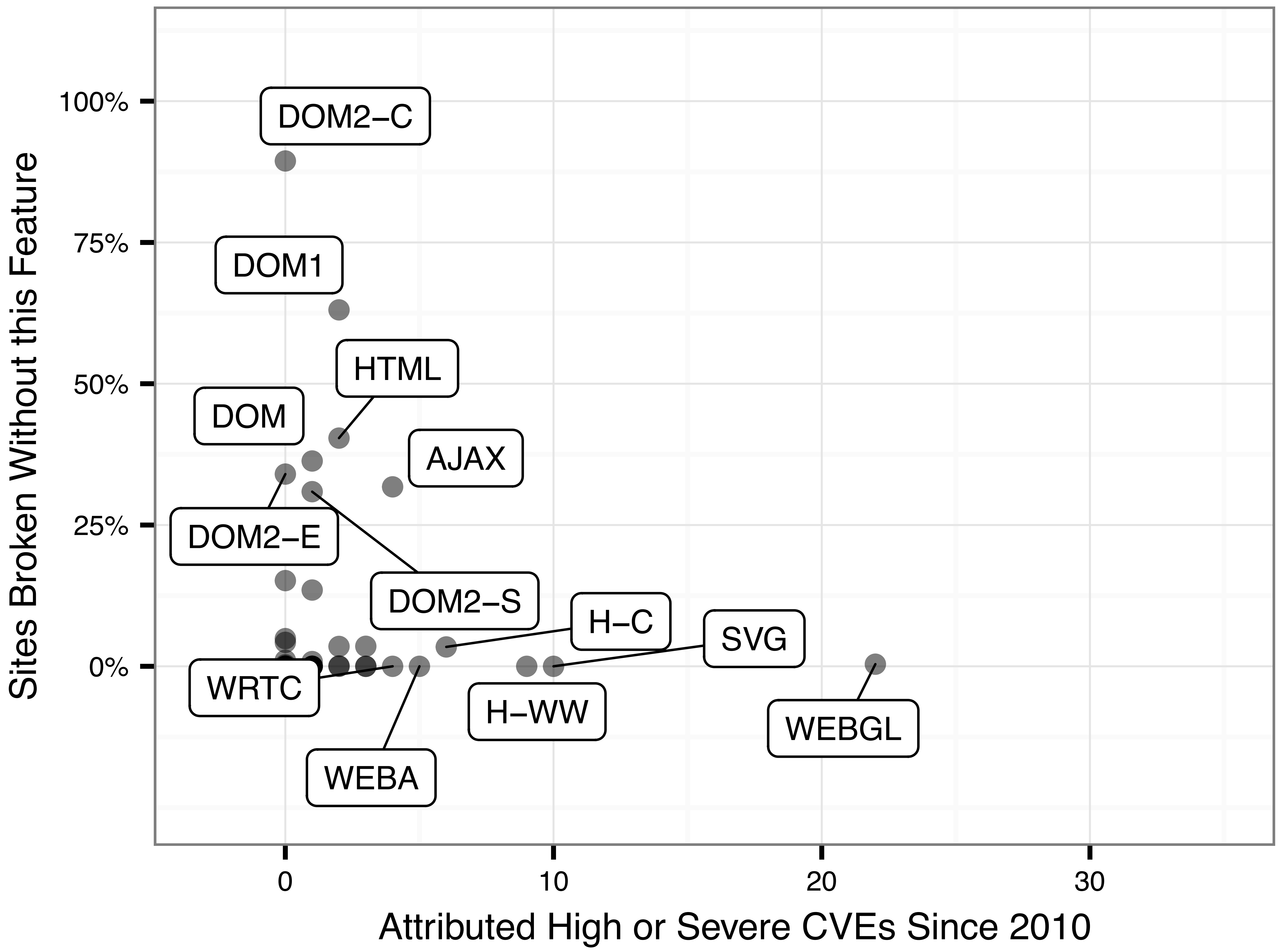
- 20 papers using 23 standards, 51 standards were never implicated
- Examples
  - **Breaking sandbox isolations with the High Resolution Timers API**
    EX: Andrysco, et al. "On subnormal floating point and abnormal timing." *S&P* 2015
  - **Fingerprinting and privacy attacks using Canvas API**
    Ex: Englehardt and Narayanan. "Online tracking: A 1-million-site measurement and analysis." *CCS* 2016
  - **Recovering length of cross origin responses using Fetch API**
    Ex: Van Goethem, et al. "Request and Conquer: Exposing Cross-Origin Resource Size." *USENIX* 2016.

Standard Cost: Related Research (2/2)

High Resolution Time Level 2	8	IEEE 2015, CCS 2015 (3), NDSS 2017, ESORICS 2015, WOOT 2014, CCS 2013
HTML: The Canvas Element	7	CCS 2014, ACSAC 2016, NDSS 2017, CCS 2016, WOOT 2014, CCS 2013, S&P 2016
Battery Status API	4	ACSAC 2016, CCS 2016, S&P 2013, Cryptology 2015
WebGL	4	ACSAC 2016, NDSS 2017, WOOT 2014, S&P 2016
Service Workers	3	CCS 2015 (2), USENIX 2016
Fetch	3	CCS 2015 (2), USENIX 2016
Web Storage	3	ACSAC 2016, WOOT 2014, CCS 2015

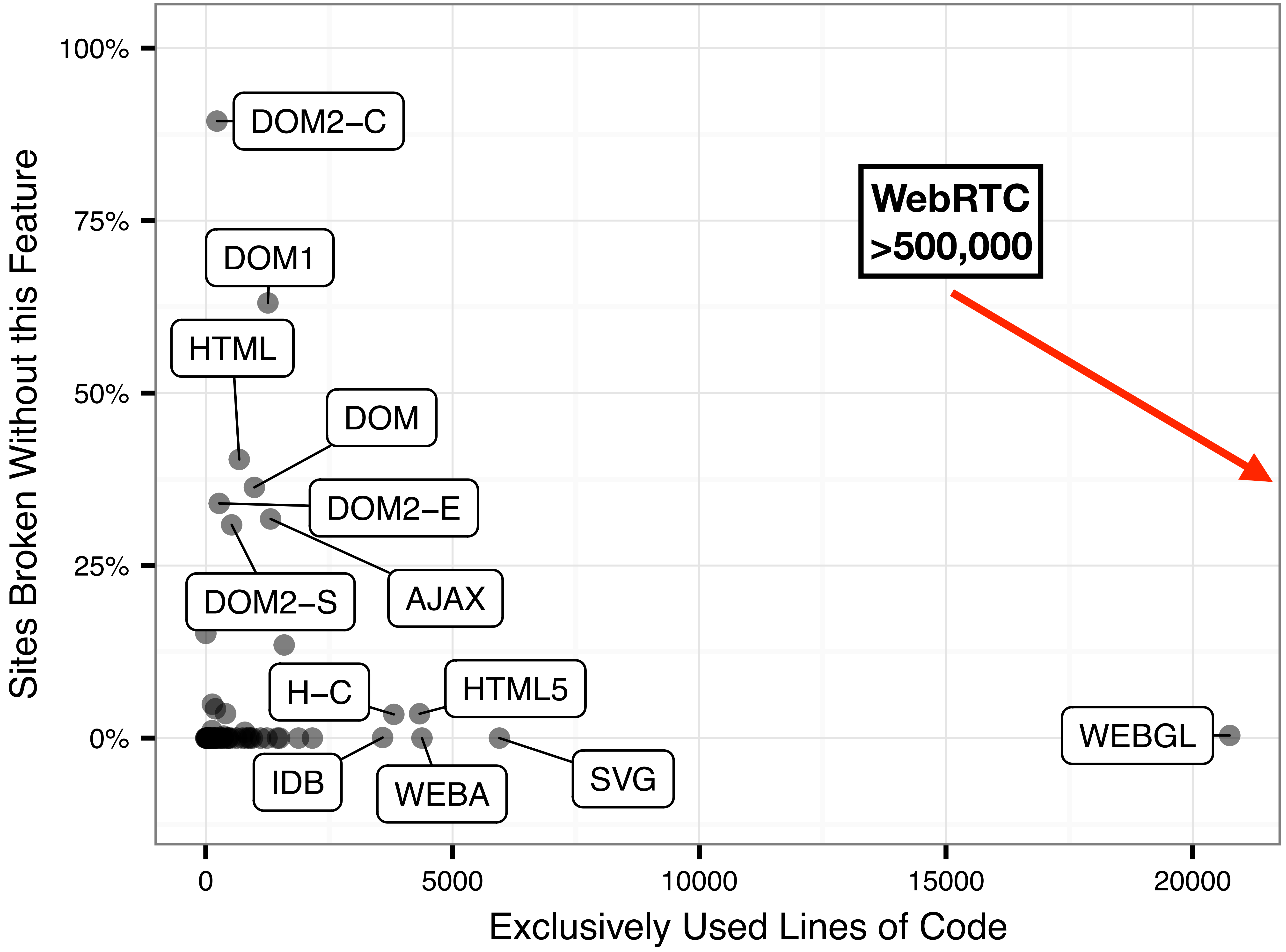
Standard Cost: CVEs

- CVEs are distributed unevenly
- A small number of Web API standards account for most CVEs since 2010
- Many frequently implicated standards are rarely used / needed
- Suggests areas for S&P benefit



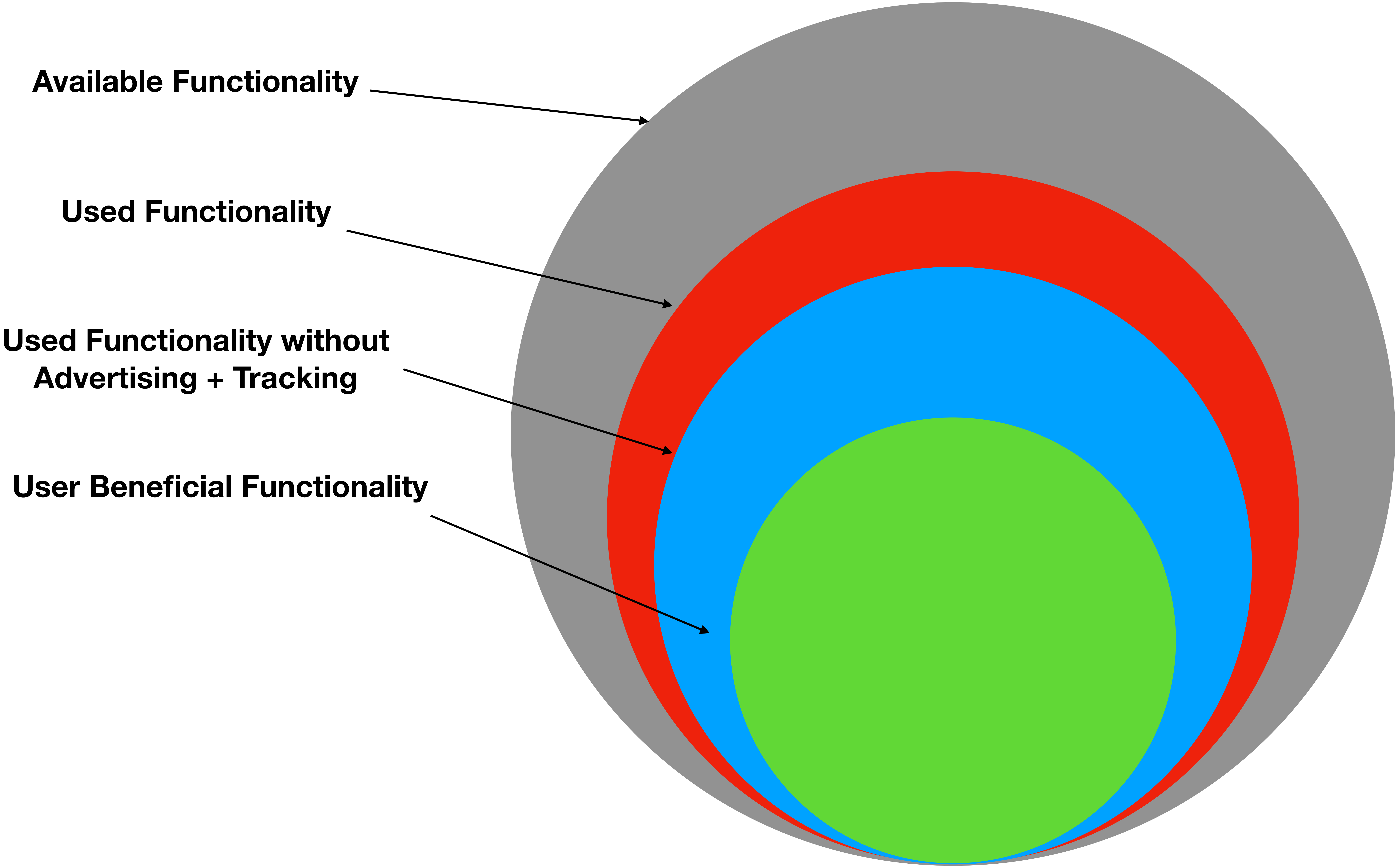
Standard Cost: Implementation Complexity

- 75,650 lines uniquely attributable
- Widely different costs between standards
- Undercounts because of:
    - third party libraries
    - shared code



Standard Name	Abbreviation	# Alexa 10k Using	Site Break Rate	Agree %	# CVEs	# High or Severe	% ELoC	Enabled attacks
WebGL	WEBGL	852	<1%	93%	31	22	27.43	[15, 21, 34, 40]
HTML: Web Workers	H-WW	856	0%	100%	16	9	1.63	[30, 34]
WebRTC	WRTC	24	0%	93%	15	4	2.48	[15, 26]
HTML: The canvas element	H-C	6935	0%	100%	14	6	5.03	[12, 15, 21, 26, 34, 38, 40]
Scalable Vector Graphics	SVG	1516	0%	98%	13	10	7.86	
Web Audio API	WEBA	148	0%	100%	10	5	5.79	[15, 26]
XMLHttpRequest	AJAX	7806	32%	82%	11	4	1.73	
HTML	HTML	8939	40%	85%	6	2	0.89	[13, 46]
HTML 5	HTML5	6882	4%	97%	5	2	5.72	
Service Workers	SW	0	0%	-	5	0	2.84	[28, 59, 60]
HTML: Web Sockets	H-WS	514	0%	95%	5	3	0.67	
HTML: History Interface	H-HI	1481	1%	96%	5	1	1.04	
Indexed Database API	IDB	288	<1%	100%	4	2	4.73	[12, 15]
Web Cryptography API	WCR	7048	4%	90%	4	3	0.52	
Media Capture and Streams	MCS	49	0%	95%	4	3	1.08	[57]
DOM Level 2: HTML	DOM2-H	8956	13%	89%	3	1	2.09	
DOM Level 2: Traversal and Range	DOM2-T	4406	0%	100%	3	2	0.04	
HTML 5.1	HTML51	2	0%	100%	3	1	1.18	
Resource Timing	RT	433	0%	98%	3	0	0.10	
Fullscreen API	FULL	229	0%	95%	3	1	0.12	
Beacon	BE	2302	0%	100%	2	0	0.23	
DOM Level 1	DOM1	9113	63%	96%	2	2	1.66	
DOM Parsing and Serialization	DOM-PS	2814	0%	83%	2	1	0.31	
DOM Level 2: Events	DOM2-E	9038	34%	96%	2	0	0.35	
DOM Level 2: Style	DOM2-S	8773	31%	93%	2	1	0.69	
Fetch	F	63	<1%	90%	2	0	1.14	[28, 59, 60]
CSS Object Model	CSS-OM	8094	5%	94%	1	0	0.17	[46]

Standard Use vs Benefit



Outline

- Background
- Measuring use
- Measuring cost vs. benefit
- <u>Applying findings to “current web”</u>
- Applying findings to “future web”

Measuring Feature Cost vs. Benefit

Snyder, Peter, Cynthia Taylor, and Chris Kanich. "Most Websites Don't Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security." *CCS, 2017*

…along with significant work conducted after publication.

Motivation from Results (1/2)

1. Web API standards differ hugely in the benefit and cost they provide browser users.

2. All standards are equally available to web sites (with rare exceptions)

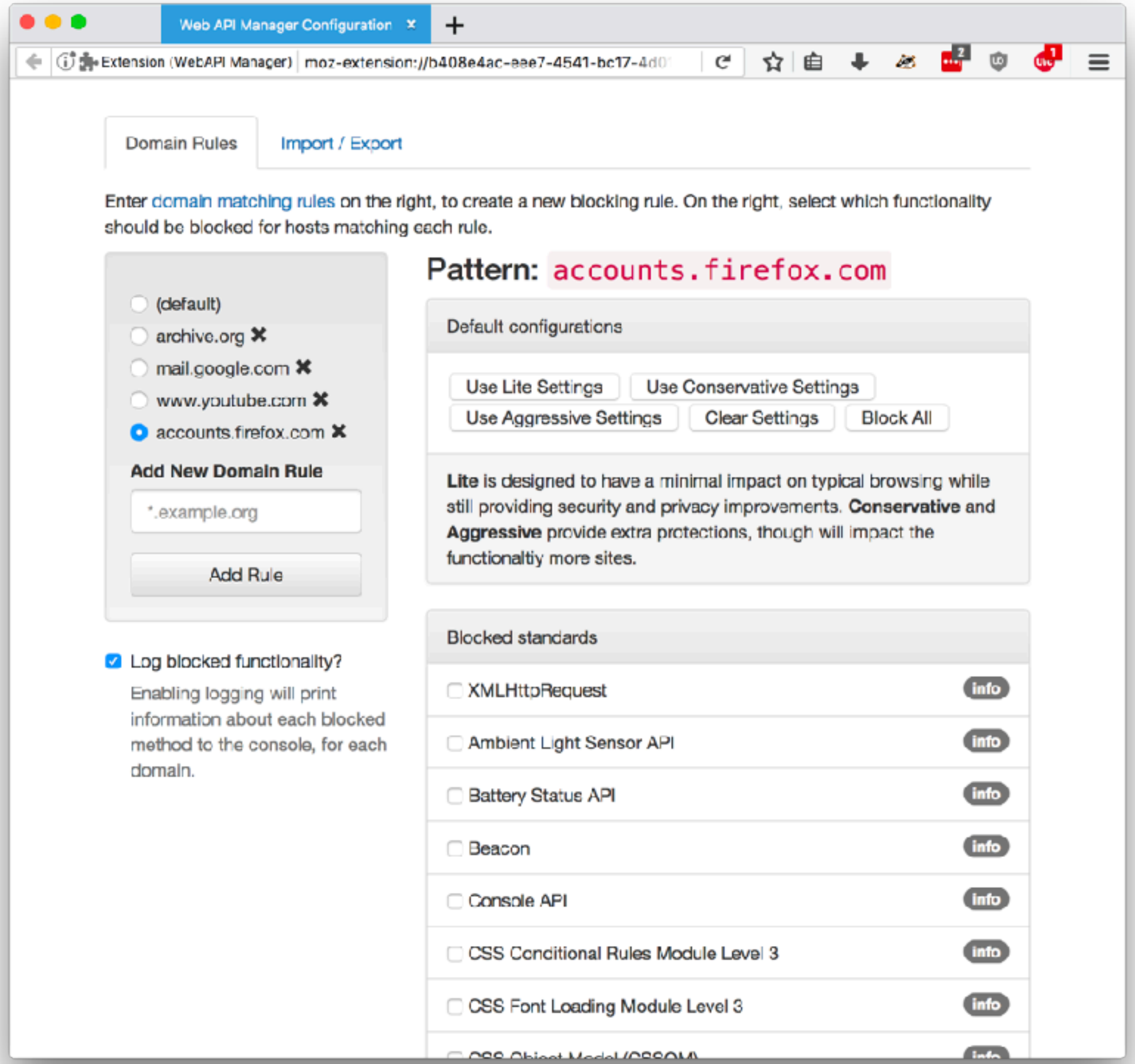
3. Users' privacy and security would be improved, at little cost, if non-trusted sites we're only given access to useful, safe features (by default).

Motivation from Results (2/2)

	Break Rate	# CVEs	# Attacks	% LOC
DOM2: Core	89%	0	0	0.29%
AJAX	32%	11	0	1.73%
Canvas	0%	13	7	5.03%
WebGL	<1%	31	4	27.43%

WebAPI Access Controls

- Browser extension that imposes access controls on Web API
- Users can restrict site access to functionality only when trusted / needed.
- Default configurations, user configurable



Usability Evaluation

- Interesting idea, but is it feasible (would anyone use it)
- Subjective measurements needed
- Impossible to evaluate 74^2 possible configurations, on all websites
- Create plausible extension configurations

Evaluated Configurations

- Two tested, realistic, configurations
- **Conservative**: Block default access to 15 rarely needed standards
- **Aggressive**: Block 45 rarely needed and / *or* high-risk standards

Standard	Conservative	Aggressive
Beacon	X	X
DOM Parsing	X	X
Full Screen	X	X
High Resolution Timer	X	X
Web Sockets	X	X
Channel Messaging	X	X
Web Workers	X	X
Index Database API	X	X
Performance Timeline	X	X
SVG 1.1	X	X
UI Events	X	X
Web Audio	X	X
WebGL	X	X
Ambient Light		X
Battery Status		X
31 more…		X

Evaluation Methodology

1. Select Representative sites
    - **Popular**: Non-pornographic, English sites in Alexa 200 (175 sites)
    - **Less Popular**: Random sampling of the rest of the Alexa 10k (155 sites)
2. Have two students visit each site for 60 seconds in default browser
3. Repeat visit in browser modified with **conservative** blocking configuration
4. Repeat visit in browser modified with **aggressive** blocking configuration
5. Compared break rates, both numerically and textually

Evaluation Findings

- Significant privacy and security benefits to blocking certain standards
- Tradeoff between S&P and functionality
- Testers agreed 97.6%-98.3% of the time

	Conservative	Aggressive
Standards Blocked	15	45
Previous CVEs Codepaths Avoided	89 (52.0%)	123 (71.9%)
LOC "Removed"	37,848 (50.00%)	37,848 (70.76%)
% Popular Sites Broken	7.14%	15.71%
% Less Popular Sites Broken	3.87%	11.61%

Usability Comparison

- How realistic are these tradeoffs?
- Repeat measurement using other popular browser privacy techniques
- Techniques compose, are not replacements

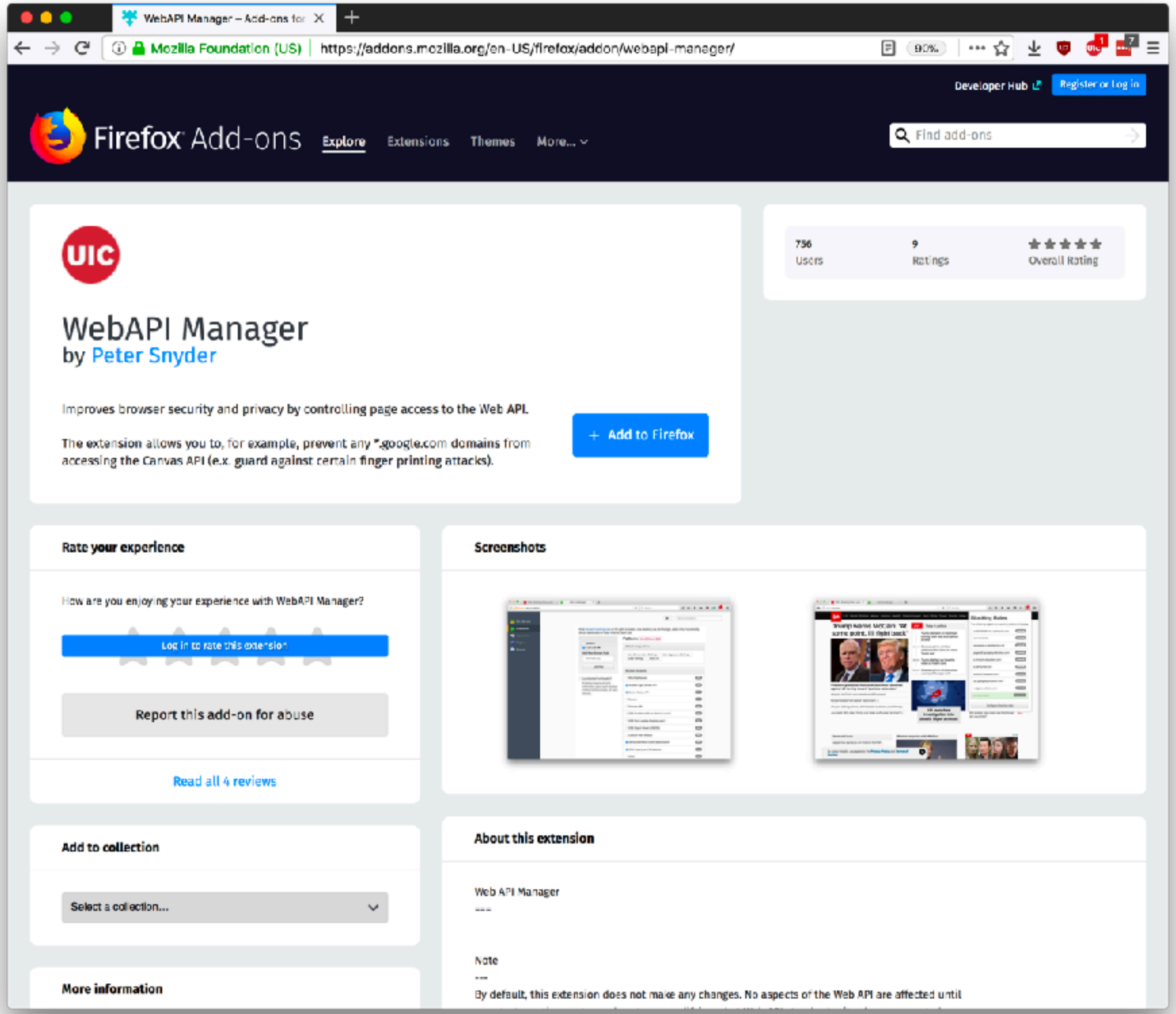
	% Popular Sites Broken	% Unpopular Sites Broken	Sites Tested
Conservative Blocking	7.14%	3.87%	330
Aggressive Blocking	15.71%	11.61%	330
Tor Browser Bundle	16.28%	7.50%	100
No Script	40.86%	43.87%	300

Improving Usability

- Moved from fixed blocking configurations to dynamic
  - Trust context aware (HTTPS, logged in, privacy modes, etc.)
  - Crowd sourced / trusted rule lists (EasyList model)
  - Third party vs. first party code
  - Dwell time
  - Single purpose applications

Lessons from Deployment

- > 1k users
- Actual, real world contributors!
- Publicity among privacy and security enthusiasts / activists
- Firefox and Chrome (and related…)
- https://github.com/snyderp/web-api-manager

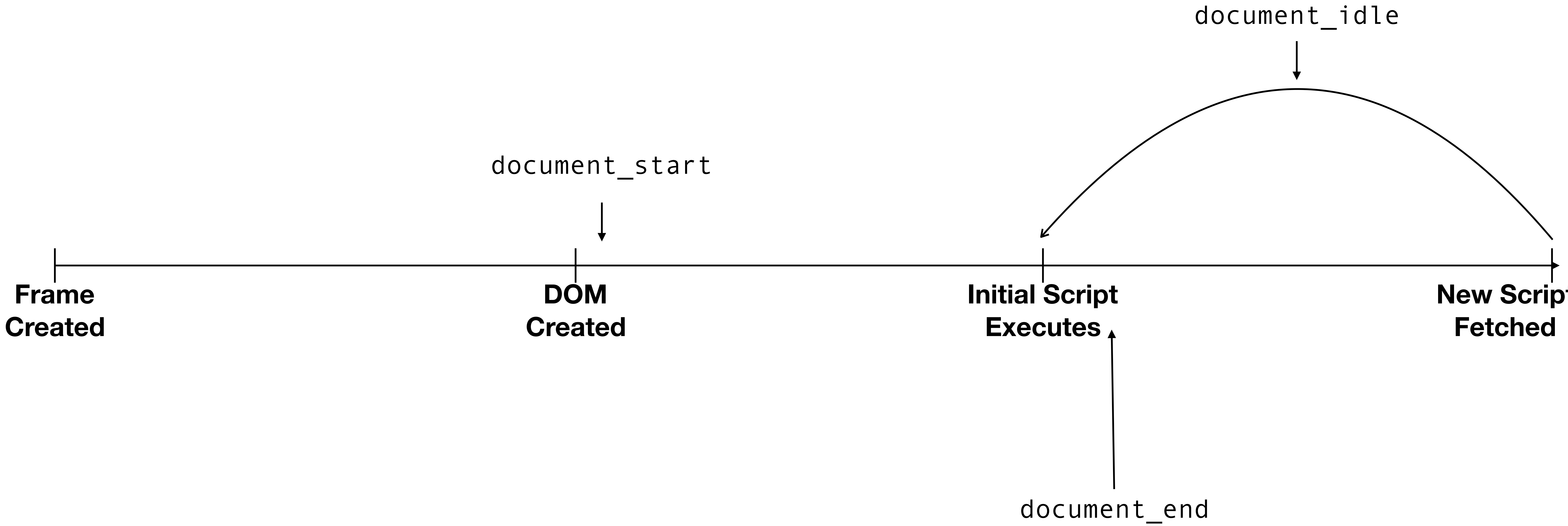


Lessons Learned from Deployment

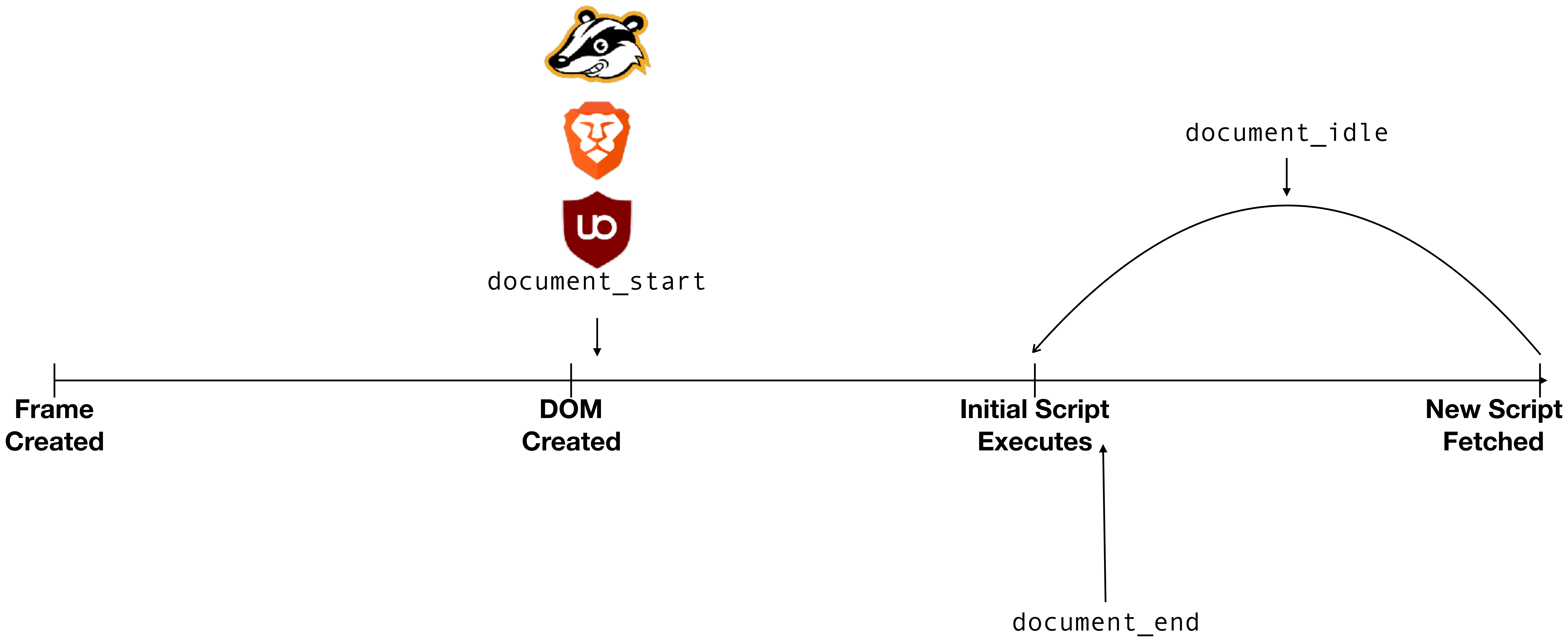
- Standards may be sub-optimal level of granularity
  - Often its just one feature (apple) that ruins the barrel
- Standards change fast
  - WebVR (2 versions!), Speech Synthesis, WebUSB, Payments API etc
- Common vulnerability in DOM modifying browser extensions

WebExtension Model

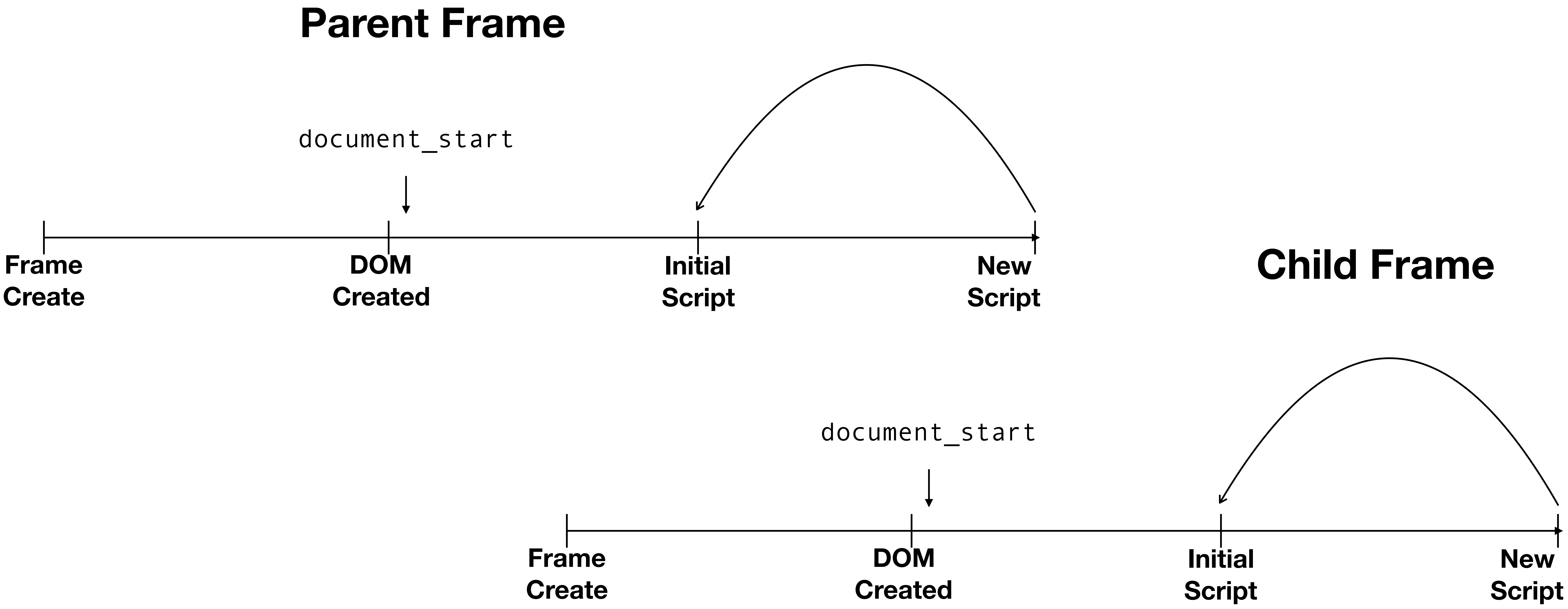
WebExtension Model



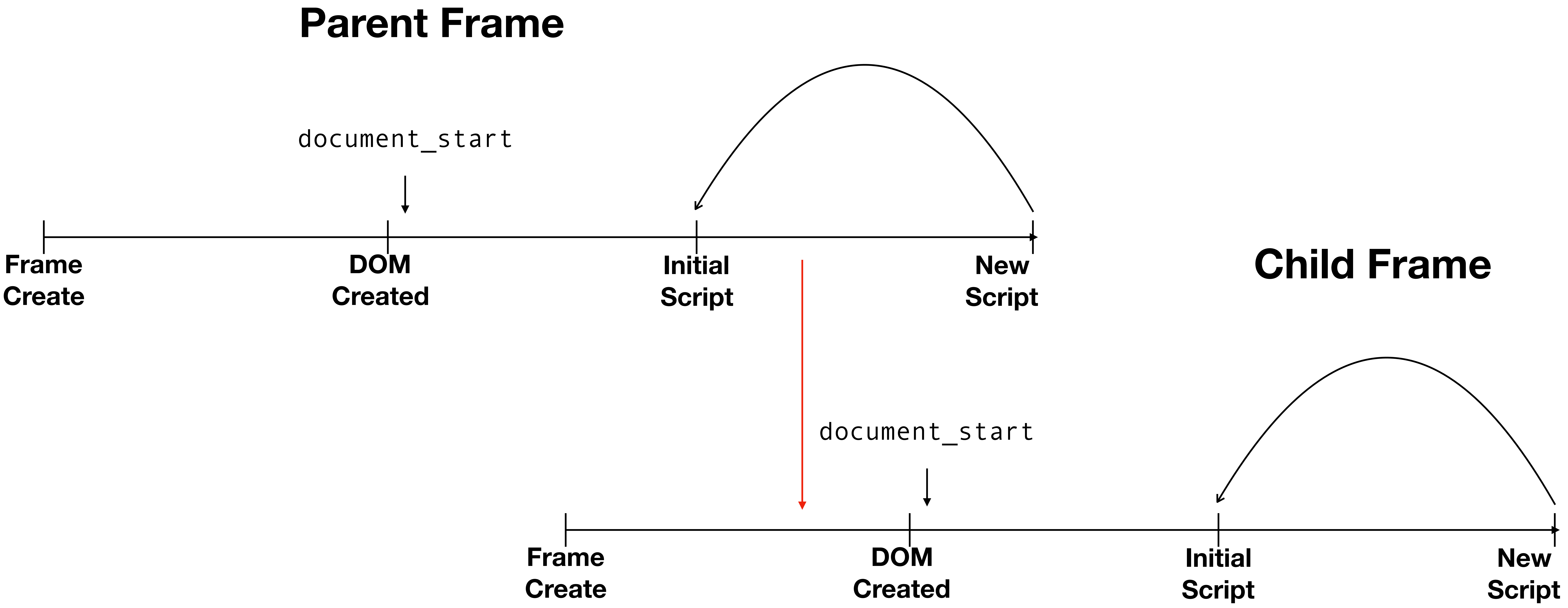
WebExtension Model



WebExtension Model



WebExtension Vulnerability



WebExtension Vulnerability

- Reported to Firefox, Chrome, Brave, EFF, uBlock Origin, etc
  - Fixed in Brave
  - Acknowledged by Firefox, EFF (Privacy Badger), and uBlock Origin
  - Still waiting in Chromium bug queue
- Possible fixes
  - Freeze parent frames while child frame is being set up
  - Move blocking into core browser functionality (TBB did, Brave now does)

Outline

- Background
- Measuring use
- Measuring cost vs. benefit
- Applying findings to “current web”
- Applying findings to “future web”

Measuring Feature Cost vs. Benefit

Snyder, Peter, Laura Watiker, Cynthia Taylor, and Chris Kanich. "CDF: Predictably Secure Web Documents." *ConPro, 2017*

…along with significant work conducted after publication.

Findings to Build On

- Most sites don't need most functionality
- Small amount of functionality gets users most of the benefits of the web
- JavaScript it difficult to predict benign from safe behavior
- Users and developers really like web application model
  - Decentralized, open, well understood application model

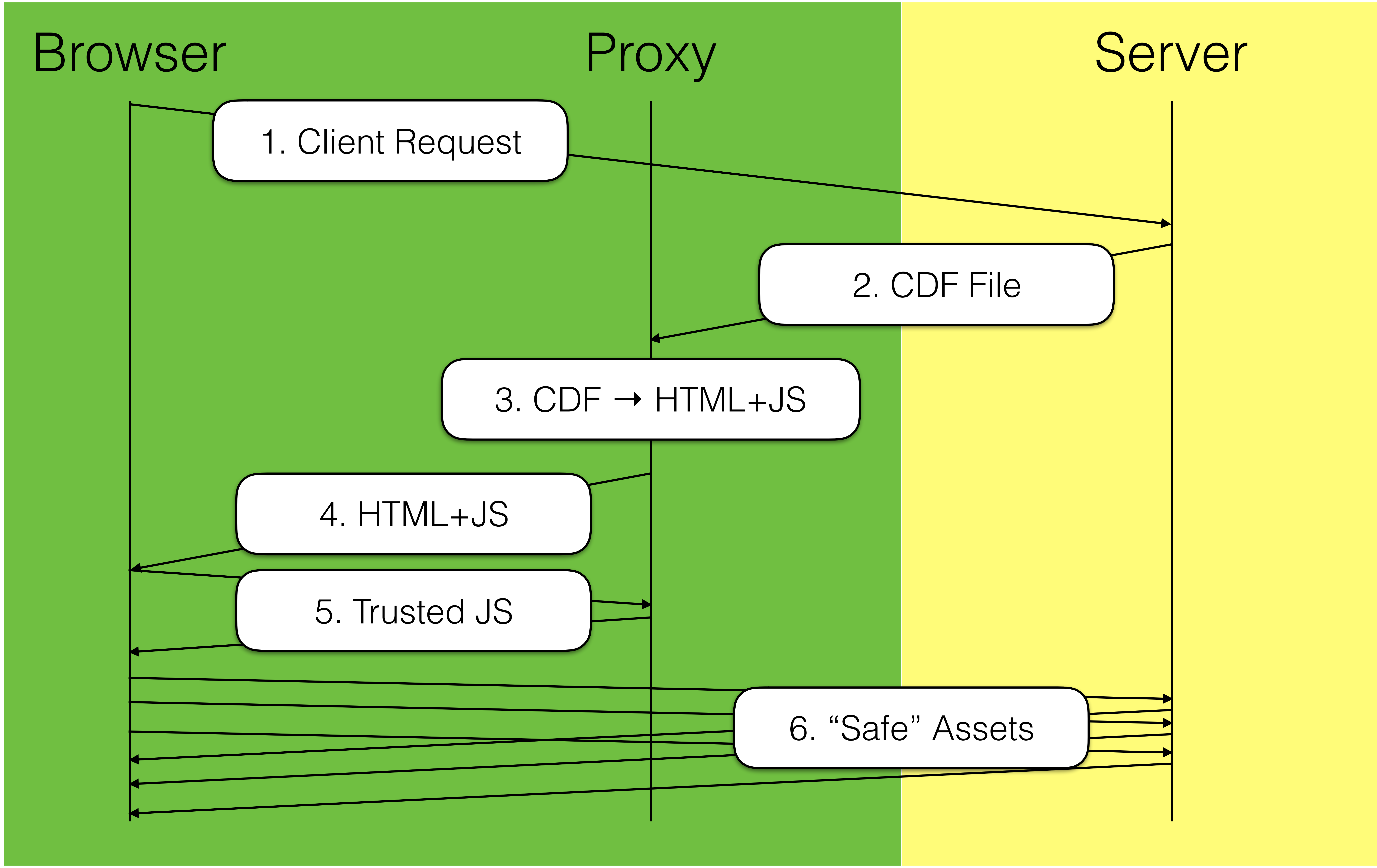
Goals for New Web Systems

- Improve privacy for non-technical users
  - Predictable, constrained information flow
- Improved security
  - Reduced attack surface, well tested code paths
- Predictable execution
  - Statically determinable execution effects

CDF Approach

Approach	Goal / Purpse
Declarative syntax	- Statically predictable behavior - Easy to write and check - Easy(er) to constan behavior
Trusted base interactive additions	- Write the tricky parts once, heavily-vet them - Allow sites to declare invocation parameters
Constrain information flow through syntax	- Disallow sending client-held information to 3rd party - Force server in the middle of third party communication
Proxy and compiler trusted based additions	- Compile statically-checked CDF into HTML+JS - Build on existing browser engineering

CDF System


Browser
Proxy
Server
1. Client Request
2. CDF File
3. CDF → HTML+JS
4. HTML+JS
5. Trusted JS
6. "Safe" Assets

CDF Example

```
{
  "t": "button",
  "c": [{"text": "click me"}],
  "e": [{
    "t": "click",
    "b": {
    "t": "states",
    "s": {
        "stateId": "text-change",
        "wrap": true,
        "states": [[[
          "button", {
            "t": "replace-sub",
            "c": {
            "text": "click on"
          }}]], [[
          "button", {
            "t": "replace-sub",
            "c": {
            "text": "click off"
}}]]]}}}]}
```

```
<button>click me</button>
<script>
let buttons = document.getElementsByTagName("button");
let stateIndex = 0;
let textStates = ["click on", "click off"];
buttons[0].addEventListener("click", function (event) {
  let newTextIndex = stateIndex++ % textStates.length;
  let newText = textStates[newTextIndex];
  event.target.innerHTML = newText;
});
</script>
```

CDF Structure

CDF Type	Purpose in System	Type Examples	Current Analogue
Structure	Define static document structure	List, List Element, Image, Video	HTML tags
Event	Define timer and user event to respond to	“timer trigger”, “mouse over”	DOM events
Behavior	Define what to do when an Event occurs	“state transition”, “remove subtree”, “change attribute”	Javascript event handlers
Delta	Define changes to the current document	“cdf sub-document”, “attribute”, “remove event”	AJAX response, WebSocket Response

System Evaluation

- **Popular blog**
  http://www.vogue.com/
- **Online-banking**
  https://www.bankofamerica.com/
- **Social media**
  https://twitter.com/
- **Collaborative web application**
  HotCRP

CDF Take Aways

- Existing system: https://github.com/bitslab/cdf
- Most of the “power” of the HTML+JS isn’t needed for most of the web
- Most of the risk of the WebAPI isn’t worth the corresponding benefit

Outline

- Background
- Measuring use
- Measuring cost vs. benefit
- Applying findings to “current web”
- Applying findings to “future web”

Conclusions

- Web is an enormously popular application system
- Web an enormously complicated system
- Its possible to evaluate the cost and benefit of discrete parts of a complicated system
- Doing so has tangible security and privacy benefits on existing systems
- Those improvements can be used to guide the design of future systems

Thank you!

Especially committee members and BITSLab comrades

(…but especially committee members)

(…but **especially** Chris :)