

Protecting Users by Confining JavaScript with COWL

Deian Stefan, Edward Z. Yang, Petr Marchenko,
Alejandro Russo, Dave Herman, Brad Karp, David Mazières



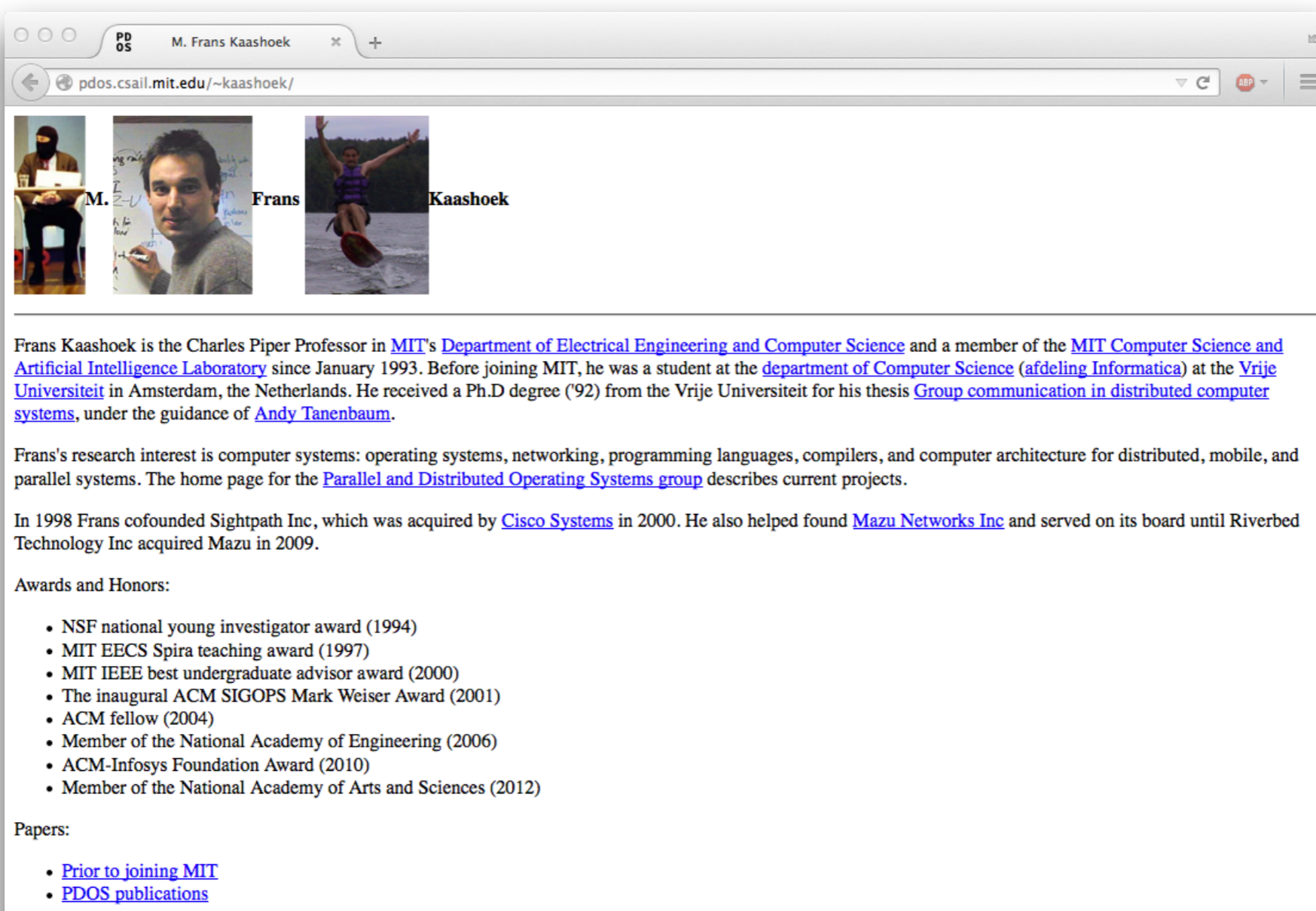
CHALMERS



mozilla

The Web

No longer just a way of publishing static content



PD OS M. Frans Kaashoek

pdos.csail.mit.edu/~kaashoek/

M. Frans Kaashoek

Frans Kaashoek

Frans Kaashoek is the Charles Piper Professor in [MIT's Department of Electrical Engineering and Computer Science](#) and a member of the [MIT Computer Science and Artificial Intelligence Laboratory](#) since January 1993. Before joining MIT, he was a student at the [department of Computer Science \(afdeling Informatica\)](#) at the [Vrije Universiteit](#) in Amsterdam, the Netherlands. He received a Ph.D degree ('92) from the Vrije Universiteit for his thesis [Group communication in distributed computer systems](#), under the guidance of [Andy Tanenbaum](#).

Frans's research interest is computer systems: operating systems, networking, programming languages, compilers, and computer architecture for distributed, mobile, and parallel systems. The home page for the [Parallel and Distributed Operating Systems group](#) describes current projects.

In 1998 Frans cofounded Sightpath Inc, which was acquired by [Cisco Systems](#) in 2000. He also helped found [Mazu Networks Inc](#) and served on its board until Riverbed Technology Inc acquired Mazu in 2009.

Awards and Honors:

- NSF national young investigator award (1994)
- MIT EECS Spira teaching award (1997)
- MIT IEEE best undergraduate advisor award (2000)
- The inaugural ACM SIGOPS Mark Weiser Award (2001)
- ACM fellow (2004)
- Member of the National Academy of Engineering (2006)
- ACM-Infosys Foundation Award (2010)
- Member of the National Academy of Arts and Sciences (2012)

Papers:

- [Prior to joining MIT](#)
- [PDOS publications](#)

The Web

Now app platform; lot of client-side functionality

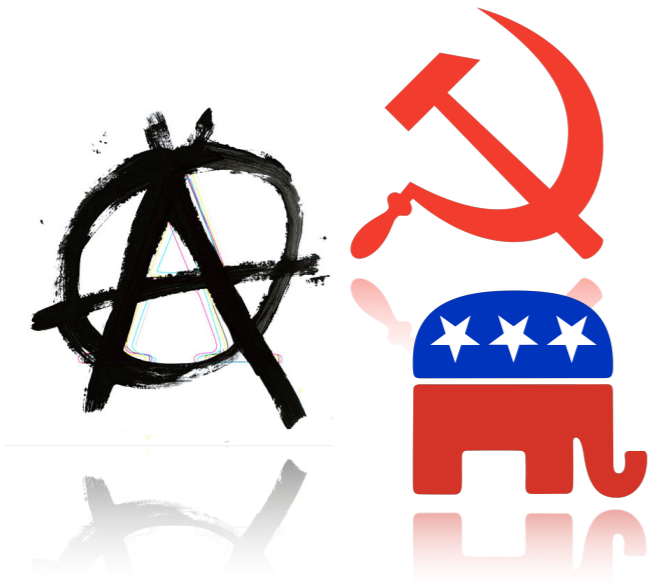
The collage displays four distinct web applications:

- Google News:** A news aggregation page with a sidebar for categories like 'Ebola', 'Syria', and 'Detroit Tigers'. The main content area features headlines such as 'Islamic State beheads British hostage Henning in new video' and 'ISIS Releases Video Purportedly Showing Beheading of Alan Henning'.
- Mint.com:** A personal finance dashboard titled 'Where You Spend' for the period 'December 1, 2007 to June 30, 2009'. It includes a pie chart showing spending distribution (e.g., 'Uncategorized Spent: \$75,692 Percent: 59%'), a table of merchant transactions, and a weather forecast for Stanford, California.
- Yelp:** A local business review platform showing a list of restaurants with their ratings and a map view of the area.
- Map Interface:** A detailed map view showing a city grid with several red location markers and a search bar.

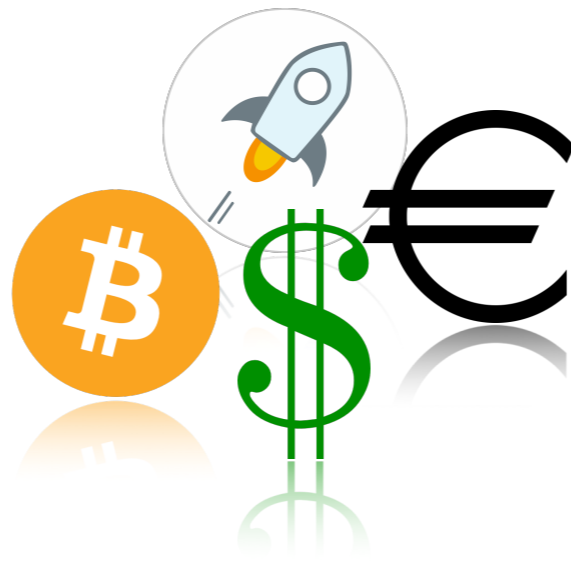
Core reason: Easy to create complex client-side apps

- Combine code and data from different parties!

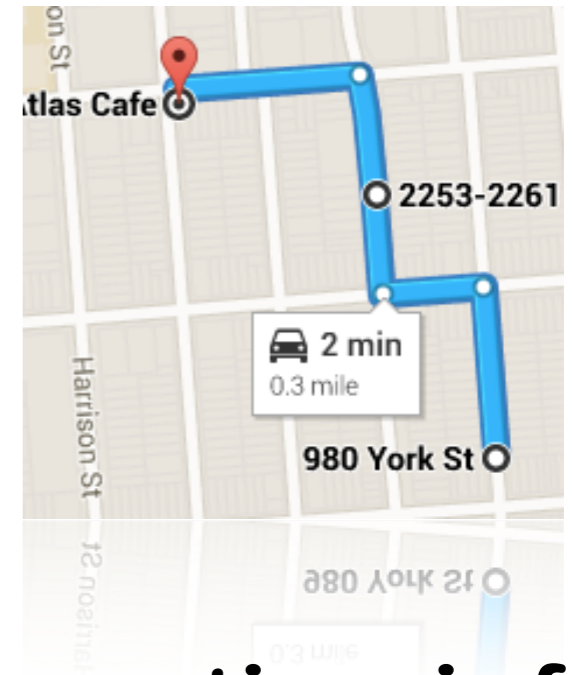
Many apps handle sensitive data



Political views



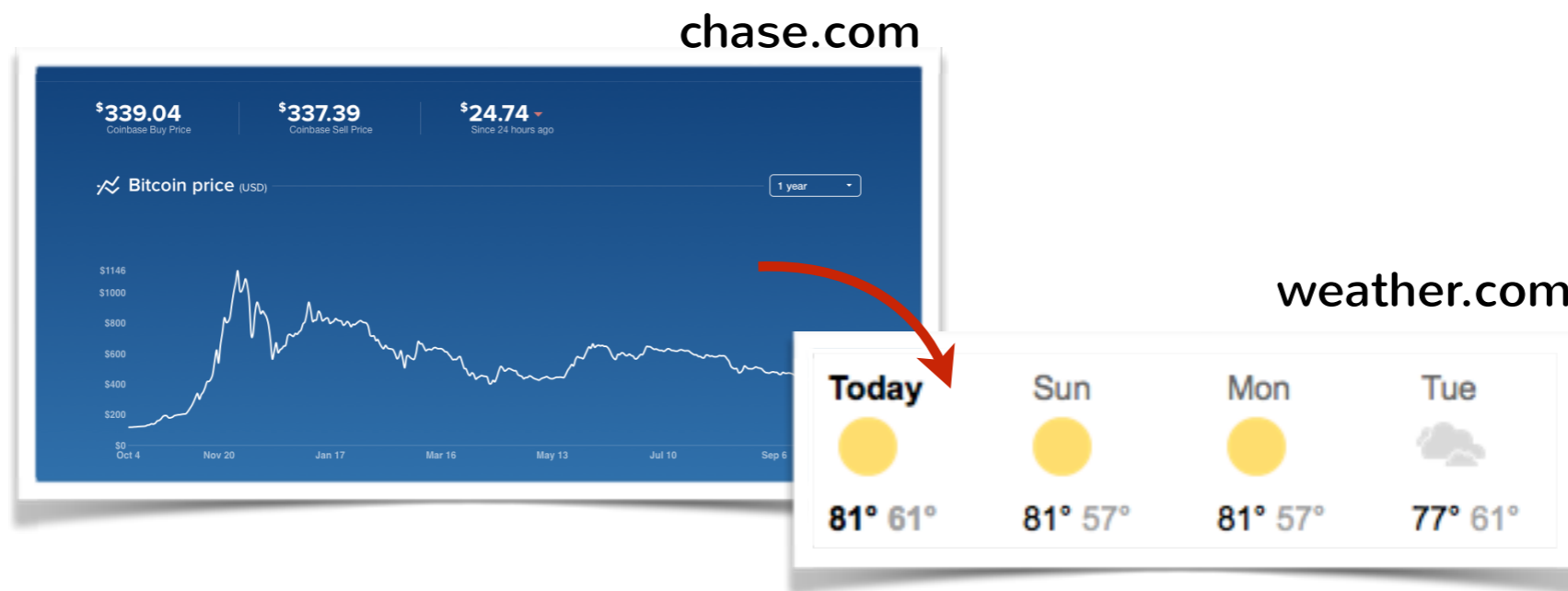
Finances



Location info

Third-party code? Sensitive data?

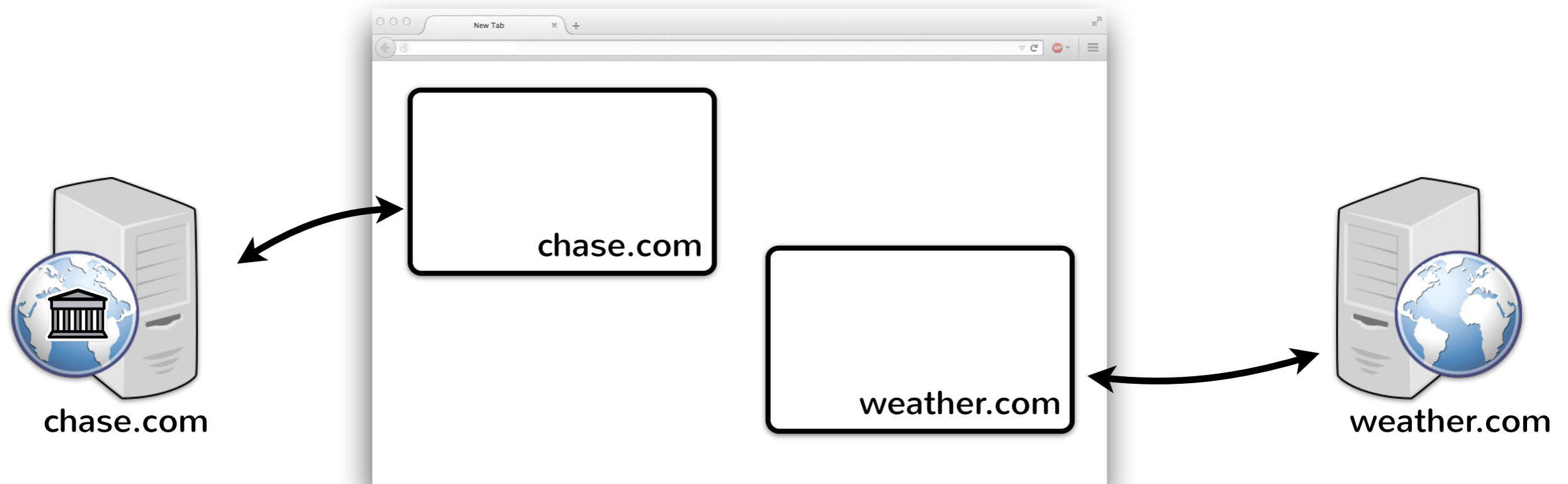
What do browsers do to ensure that the weather site cannot access my bank statements?



In the beginning: Same-origin Policy

Idea: isolate content from different origins

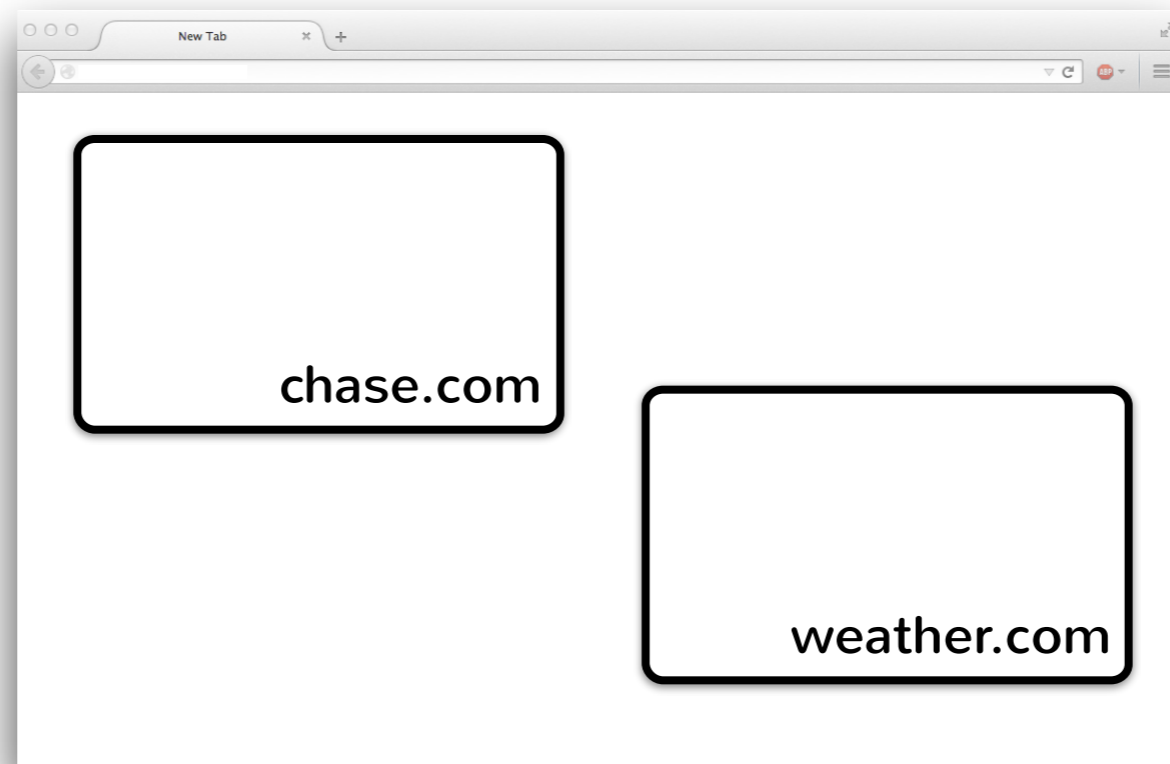
- Compartmentalize code into contexts (tabs, iframes,...)
- Disallow cross-origin reads from contexts & servers



In the beginning: Same-origin Policy

Idea: isolate content from different origins

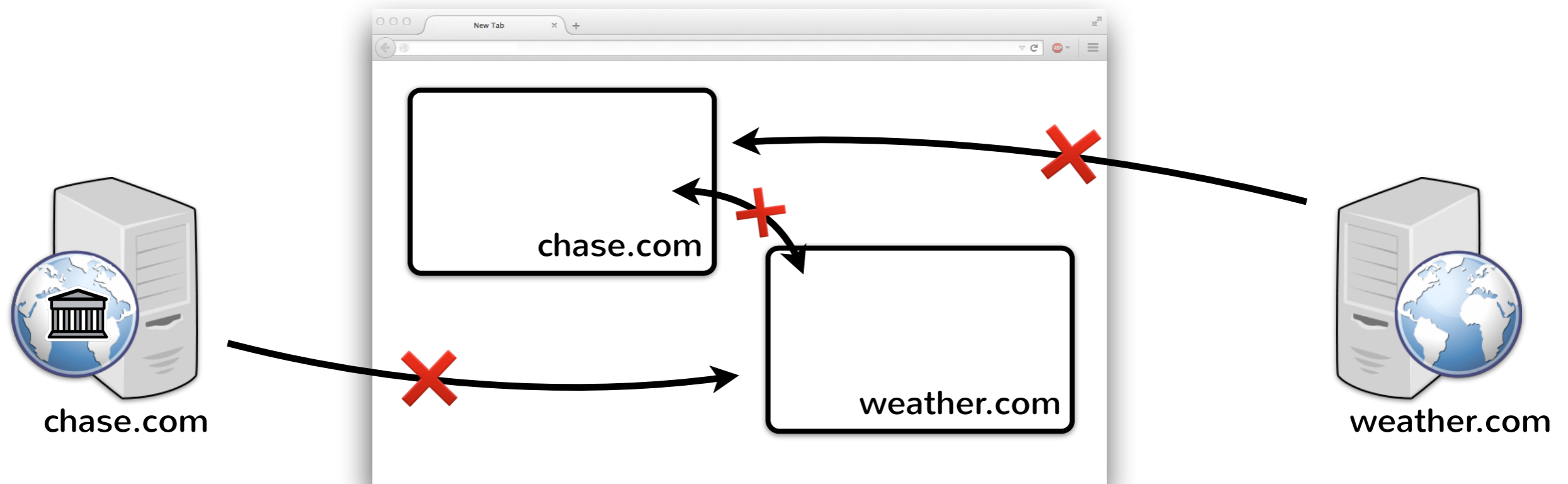
- Compartmentalize code into contexts (tabs, iframes,...)
- Disallow cross-origin reads from contexts & servers



In the beginning: Same-origin Policy

Idea: isolate content from different origins

- Compartmentalize code into contexts (tabs, iframes,...)
- Disallow cross-origin reads from contexts & servers



Problems with SOP

Not strict enough:

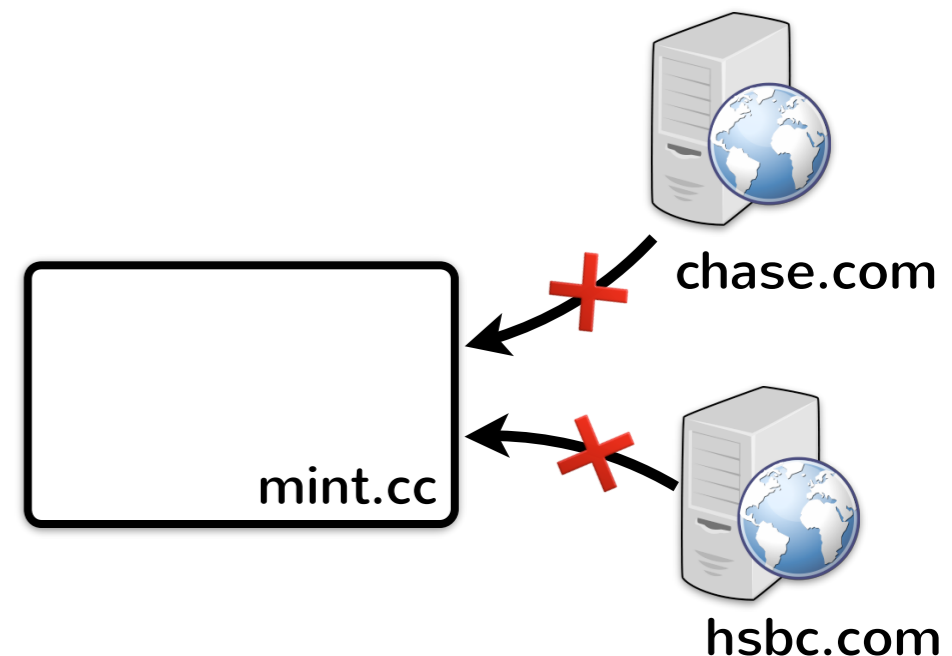
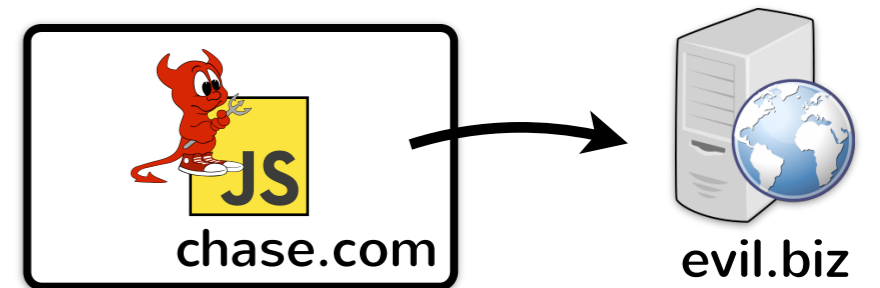
can disclose data arbitrarily

- Third-party code can leak data
- Code runs with authority of page

Not flexible enough:

can't read cross-origin data

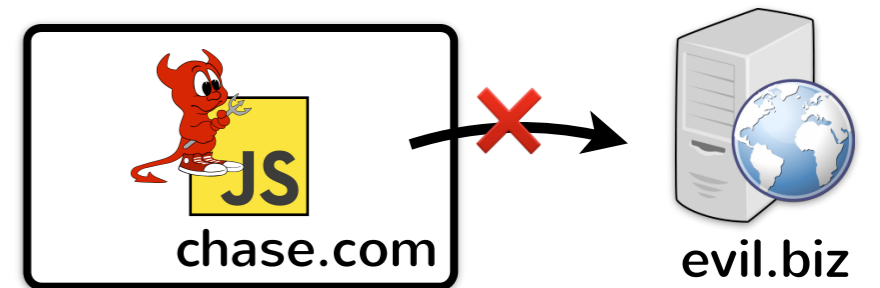
- No secure third-party mashups!



Today: SOP + CSP + CORS

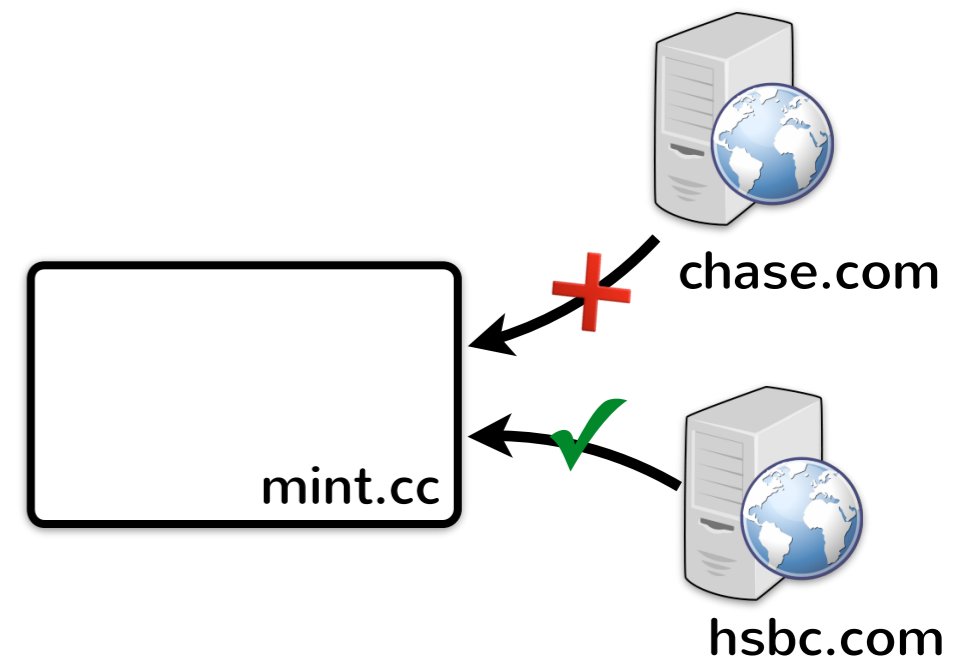
Content Security Policy:

- Whitelist origins page can communicate with



Cross-origin Resource Sharing:

- Server whitelists origins allowed to read the data



Today: SOP + CSP + CORS

Content Security Policy:

- Whitelist origins page can communicate with

Cross-origin Resource Sharing:

- Server whitelists origins allowed to read the data



**Discretionary
Access
Control**

DAC is not enough!

Forces choice between functionality and privacy

- E.g., mint.com-like client-side third-party mashup



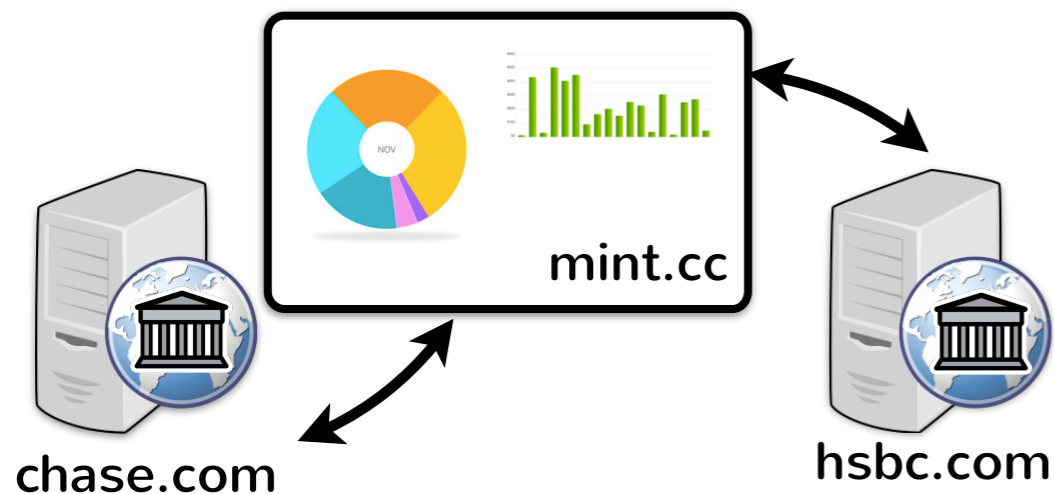
- **Privacy:** bank doesn't give mint.cc access to data
- **Functionality:** bank cedes user data to mint.cc
(or worse: user cedes bank credentials)

DAC is not enough!

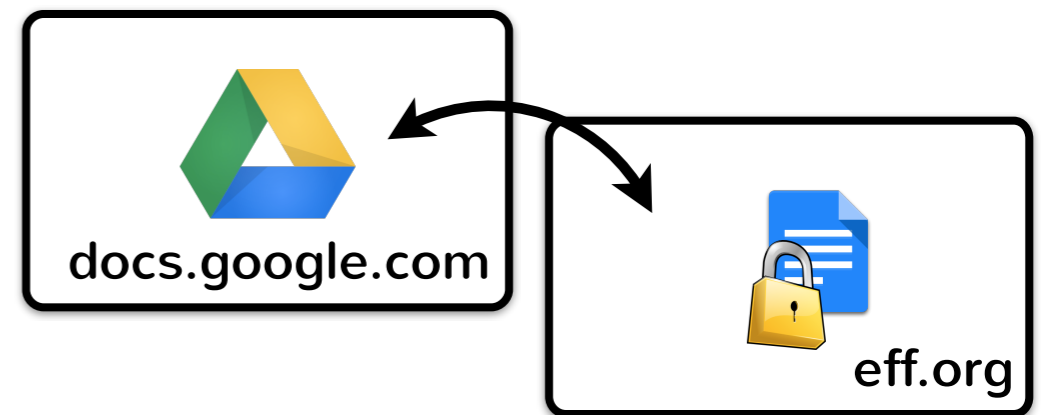
Reality: we give up privacy for functionality!

DAC is not enough!

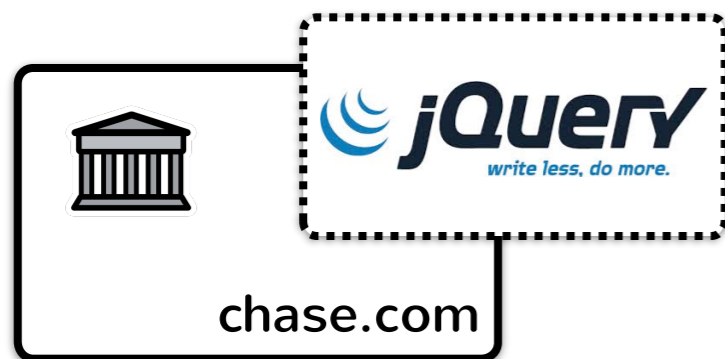
Third-party mashups



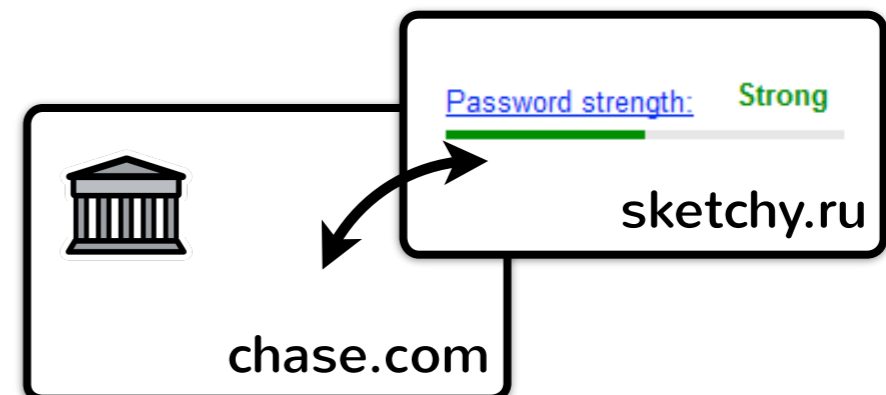
Mutually distrusting services



Tightly-coupled libraries



Libraries with narrow APIs



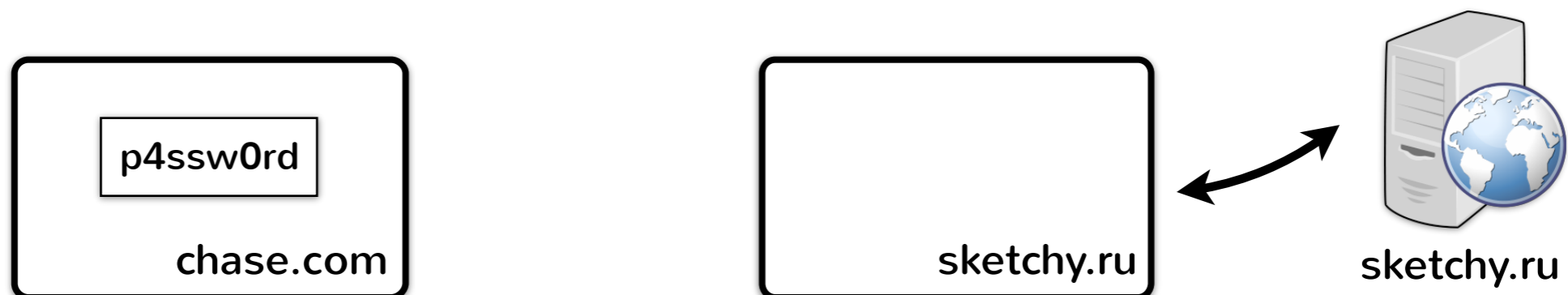
Third-party code + sensitive data

Challenge: allow untrusted code to compute on data

- E.g., chase wants to use password-strength checker library needs to fetch list of common passwords
 - Safe to fetch list before looking at password!

Need: confinement (MAC)

- Impose restrictions on how code uses data



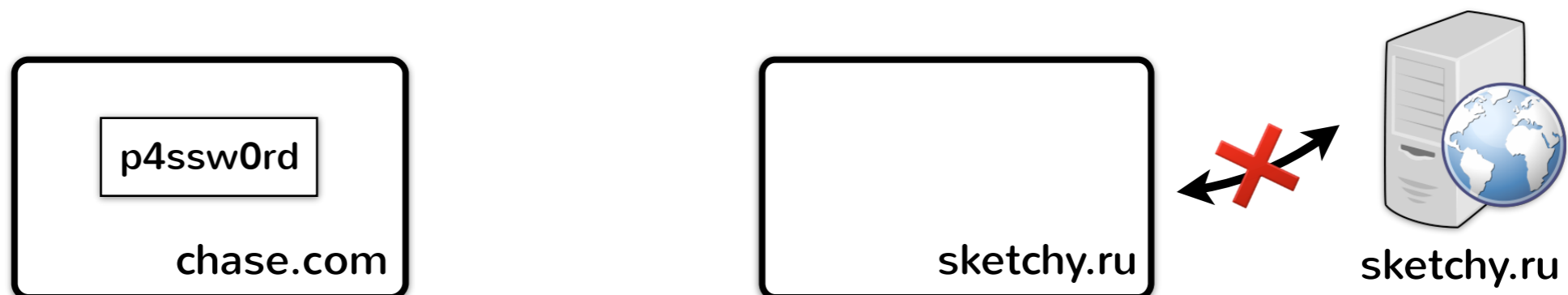
Third-party code + sensitive data

Challenge: allow untrusted code to compute on data

- E.g., chase wants to use password-strength checker library needs to fetch list of common passwords
 - Safe to fetch list before looking at password!

Need: confinement (MAC)

- Impose restrictions on how code uses data



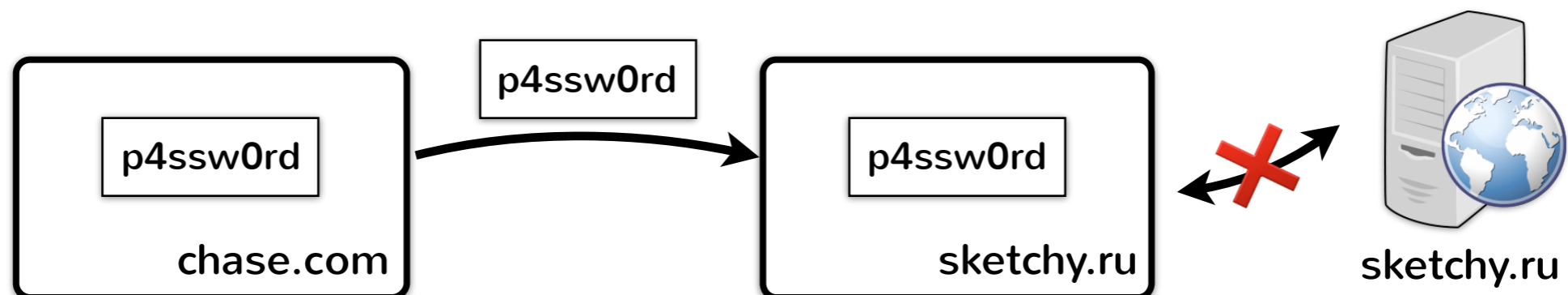
Third-party code + sensitive data

Challenge: allow untrusted code to compute on data

- E.g., chase wants to use password-strength checker library needs to fetch list of common passwords
 - Safe to fetch list before looking at password!

Need: confinement (MAC)

- Impose restrictions on how code uses data



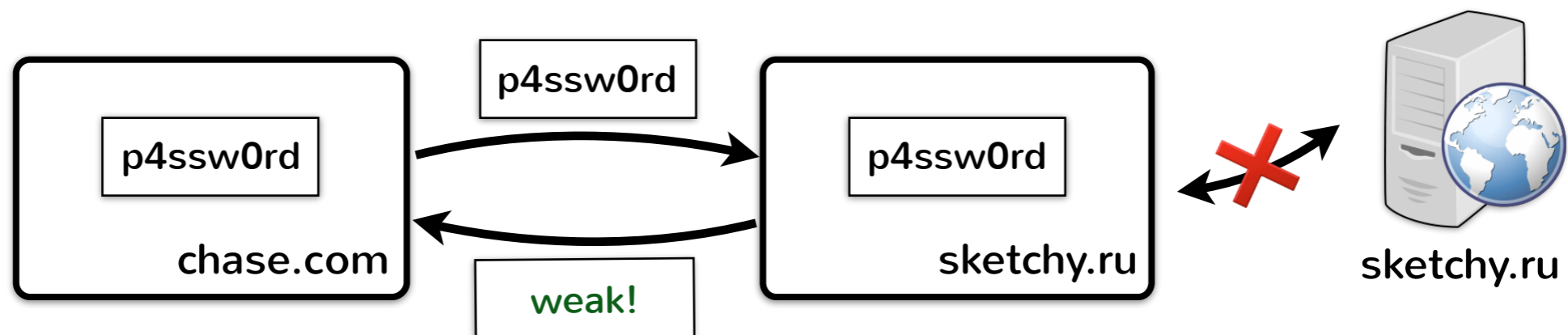
Third-party code + sensitive data

Challenge: allow untrusted code to compute on data

- E.g., chase wants to use password-strength checker library needs to fetch list of common passwords
 - Safe to fetch list before looking at password!

Need: confinement (MAC)

- Impose restrictions on how code uses data



Isn't confinement a solved problem?

Confinement for Haskell Hails

```
collection "papers" $ do
  access $ do
    readers ==> anybody
    writers ==> anybody
  clearance $ do
    secrecy ==> this
    integrity ==> anybody
```

Confinement for Java Jif!

```
ElectionCache{voter<-voter} cache,
Ballot [{voter->voter;voter<-voter}] {voter<-voter} b,
Map [{voter->voter;voter<-voter}, {voter->voter;voter<-voter}] {vote
IllegalArgumentException{}} where caller(voter) {
  if (details == null || details.ballotDesign == null ||
      details.electionID == null || b == null || caps == null) return
```

Change JavaScript to enforce IFC with JSFlow


$$\begin{array}{c}
 \frac{GetOwnProperty(\dot{r}, \dot{s}) \ E = \{value \mapsto \dot{v}\}}{GetProperty(\dot{r}, \dot{s}) \ E = \{value \mapsto \dot{v}\}} \quad \frac{GetOwnProperty(\dot{r}, \dot{s}) \ E = undefined^{\sigma_1} \quad o = E[r] \quad o[_{Prototype}] = null^{\sigma_2}}{GetProperty(\dot{r}, \dot{s}) \ E = undefined^{\sigma_1 \sqcup \sigma_2}} \\
 \frac{o = E[r_1] \quad o[_{Prototype}] = \dot{r}_2 \quad GetProperty(\dot{r}_2, \dot{s}) \ E = d}{GetProperty(\dot{r}_1, \dot{s}) \ E = d^{\sigma_1}} \quad \frac{GetProperty(\dot{r}, \dot{s}) \ E = \{value \mapsto p^\sigma\}}{HasProperty(\dot{r}, \dot{s}) \ E = true^\sigma} \\
 \frac{GetProperty(\dot{r}, \dot{s}) \ E = undefined^\sigma}{HasProperty(\dot{r}, \dot{s}) \ E = false^\sigma} \quad \frac{o = \phi[r] \quad o = \{s \xrightarrow{\sigma_3} \dot{v}, \dots, \sigma_4\} \quad \sigma = pc \sqcup \epsilon \sqcup \sigma_1 \quad \sigma <: \sigma_4 \quad \sigma \sqcup \sigma_2 <: \sigma_3}{pc \vdash Delete(r^{\sigma_1}, s^{\sigma_2}) \ (\phi, \epsilon) = (true^L, (\phi[r \mapsto o \setminus s, \sigma_2 \sqcup \sigma_4], \epsilon))} \\
 \frac{o = E[r] \quad s \notin dom(o)}{pc \vdash Delete(\dot{r}, \dot{s}) \ E = (true^L, E)} \quad \frac{GetProperty(\dot{r}, \dot{s}) \ E = \{value \mapsto \dot{v}\}}{Get(\dot{r}, \dot{s}) \ E = \dot{v}} \quad \frac{GetProperty(\dot{r}, \dot{s}) \ E = undefined^\sigma}{Get(\dot{r}, \dot{s}) \ E = undefined^\sigma} \\
 \frac{o_1 = \phi[r] \quad o_1 = \{s \xrightarrow{\sigma_3} v_2^{\sigma_4}, \dots, \sigma_5\} \quad \sigma = pc \sqcup \epsilon \sqcup \sigma_1 \quad \sigma \sqcap \sigma_2 <: \sigma_5 \quad \sigma \sqcup \sigma_2 <: \sigma_4 \quad o_2 = o_1[s \xrightarrow{(\sigma \sqcup \sigma_2) \sqcap \sigma_3} \dot{v}_1^{\sigma \sqcup \sigma_2}, \sigma_2 \sqcup \sigma_5]}{pc \vdash Put(r^{\sigma_1}, s^{\sigma_2}, \dot{v}_1) \ (\phi, \epsilon) = (\phi[r \mapsto o_2], \epsilon)} \quad \frac{o_1 = \phi[r] \quad s \notin dom(o_1) \quad o_1 = \{\dots, \sigma_3\} \quad \sigma = pc \sqcup \epsilon \sqcup \sigma_1 \quad \sigma <: \sigma_3 \quad o_2 = o_1[s \xrightarrow{\sigma \sqcup \sigma_2} \dot{v}_1^{\sigma \sqcup \sigma_2}, \sigma_2 \sqcup \sigma_3]}{pc \vdash Put(r^{\sigma_1}, s^{\sigma_2}, \dot{v}_1) \ (\phi, \epsilon) = (\phi[r \mapsto o_2], \epsilon)}
 \end{array}$$

```
ails, cache);
icKey(details
oter});
```

Dev...



Design constraints

- Can't expect developers to learn new language
- Can't touch JavaScript runtime
 - Highly optimized JITs
 - Add 1 instruction on hot path  no upstream!
- Can't radically change the security model
 - Ingrained notion of principals: origins
 - Keep iframes, pages, etc. as security boundaries

The good news

By accident...

Web turns out to be a good fit for confinement

...if you just look at it right

The good news

- **Browsers already offer execution contexts**
 - Isolation enforced across context boundaries
- **Can enforce MAC at context granularity**
 - No need to change language runtime! [BFlow]
- **Can easily add new DOM-level APIs**
 - Attach policies to messages [Hails]

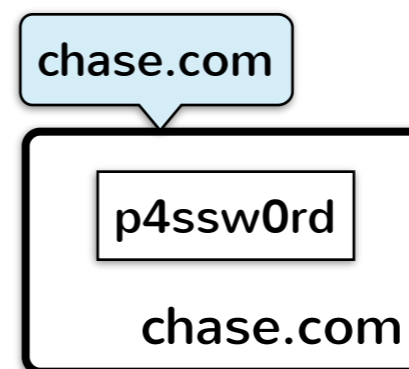
Confinement with Origin Web Labels (COWL)

Key (old) concepts: expressed in practical way?

1. **Labels:** using origins to specify MAC policies
2. **Labeled communication:** security across contexts
 - Avoid changing existing communication APIs
3. **Privileges:** using origins to manage trust

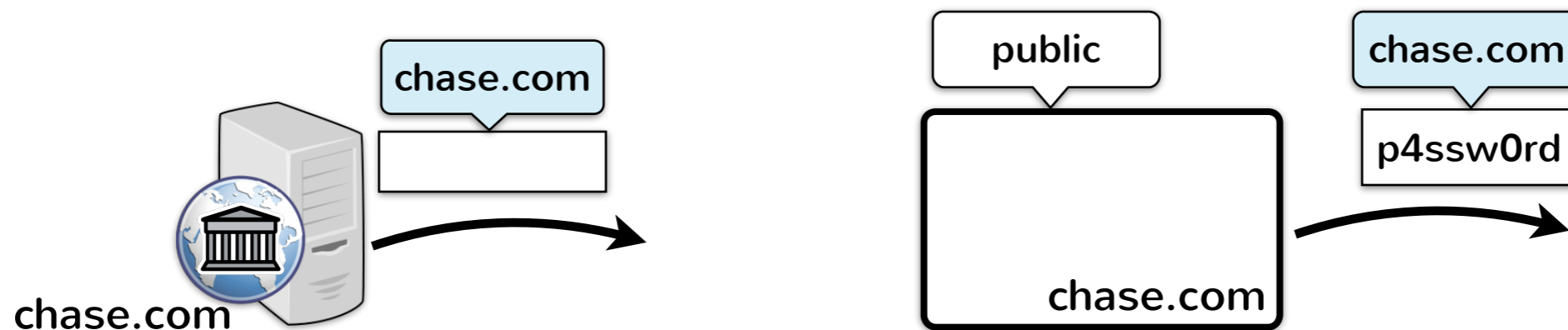
Labels

- Every piece of data is protected by a label
- Label specifies, in terms of origin(s), who cares about the data
 - E.g., data sensitive to Chase: `Label("chase.com")`
 - E.g., data sensitive to both Chase and HSBC: `Label("chase.com").and("hsbc.com")`



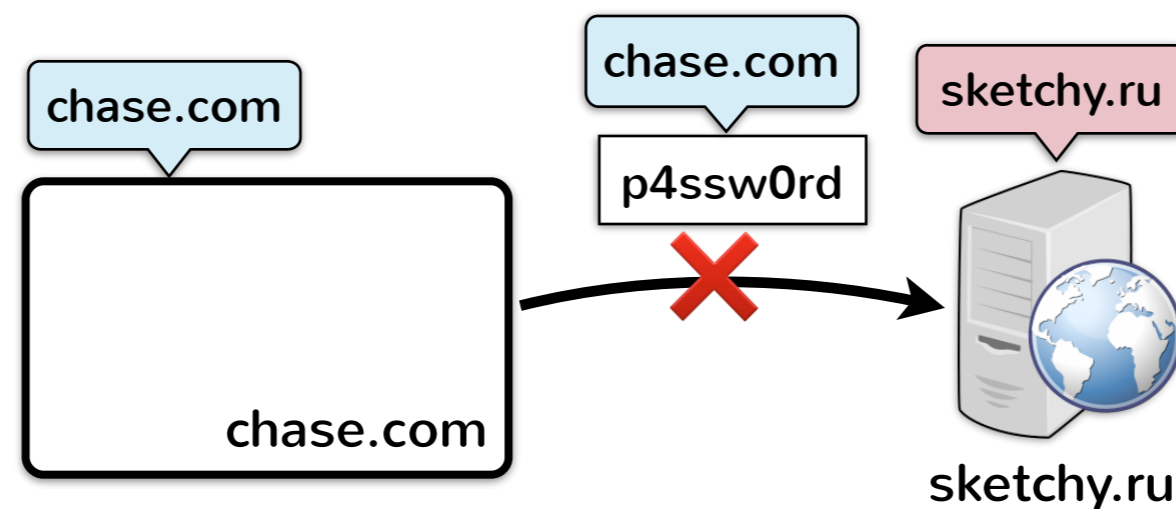
Label tracking

- COWL tracks labels at context/server granularity
 - Pages, iframes, workers, servers
- Messages can be labeled differently from context
 - Both servers & JavaScript can label messages
 - The right way to share sensitive data!



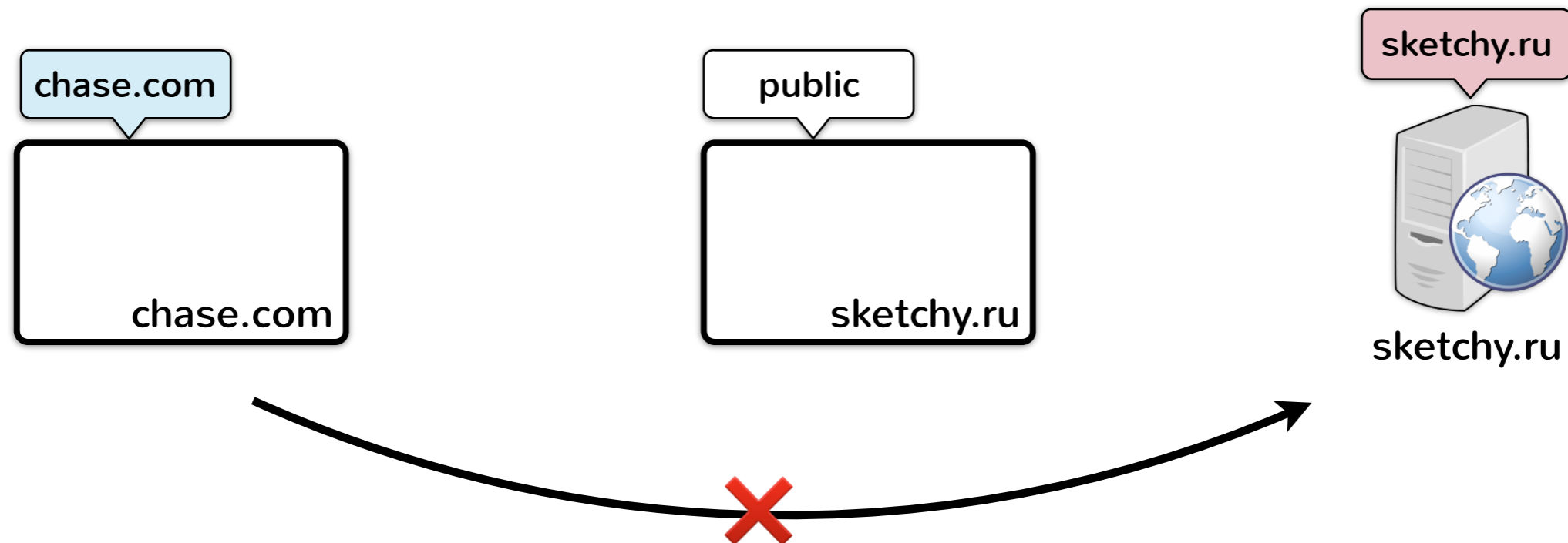
Labeled Communication

- Browser-server communication must respect labels!



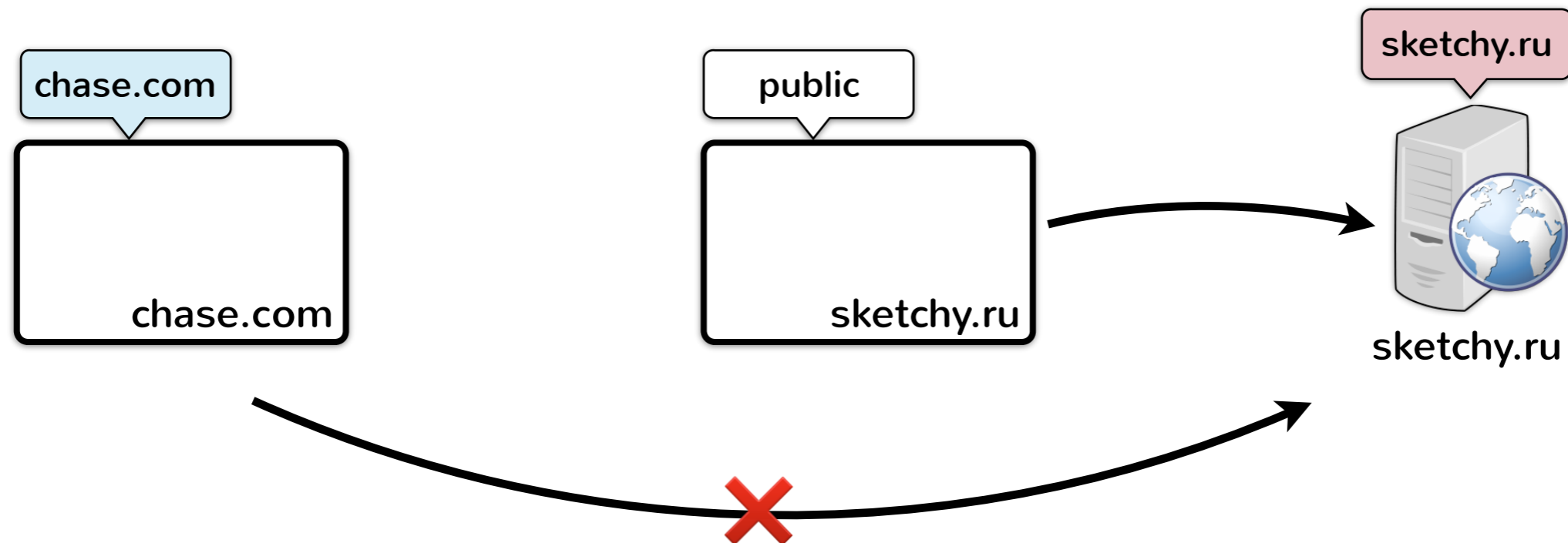
Labeled Communication

- Communication across browser contexts must respect label



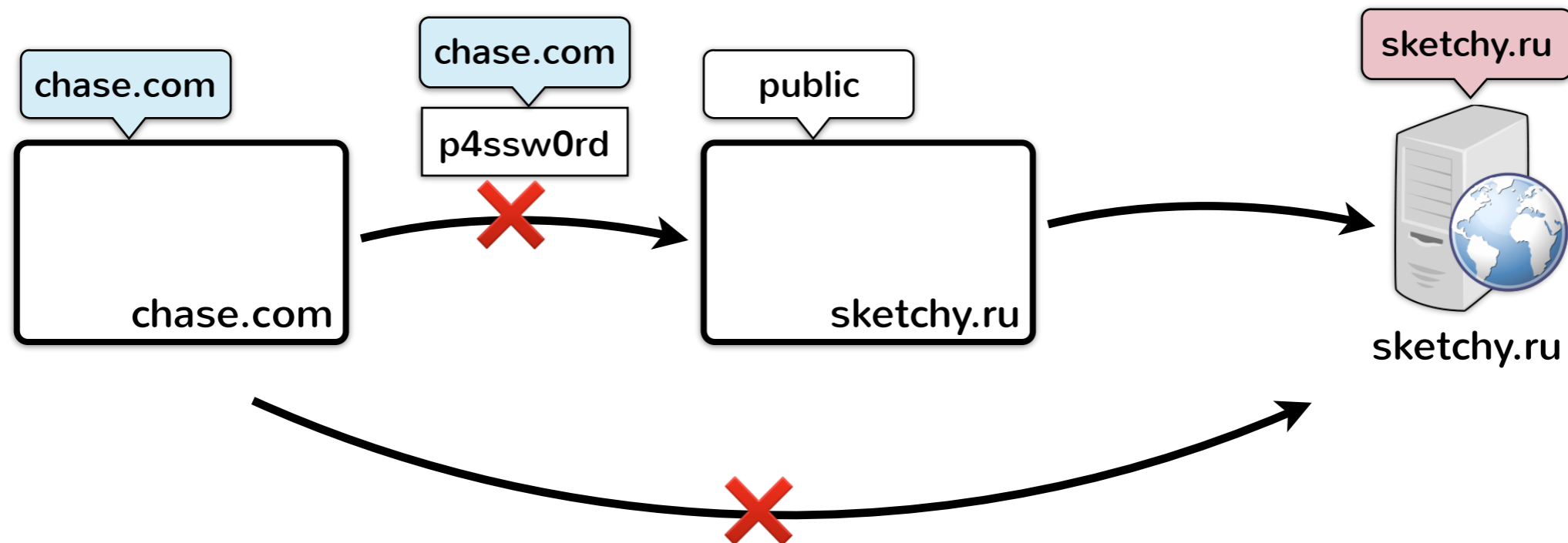
Labeled Communication

- Communication across browser contexts must respect label



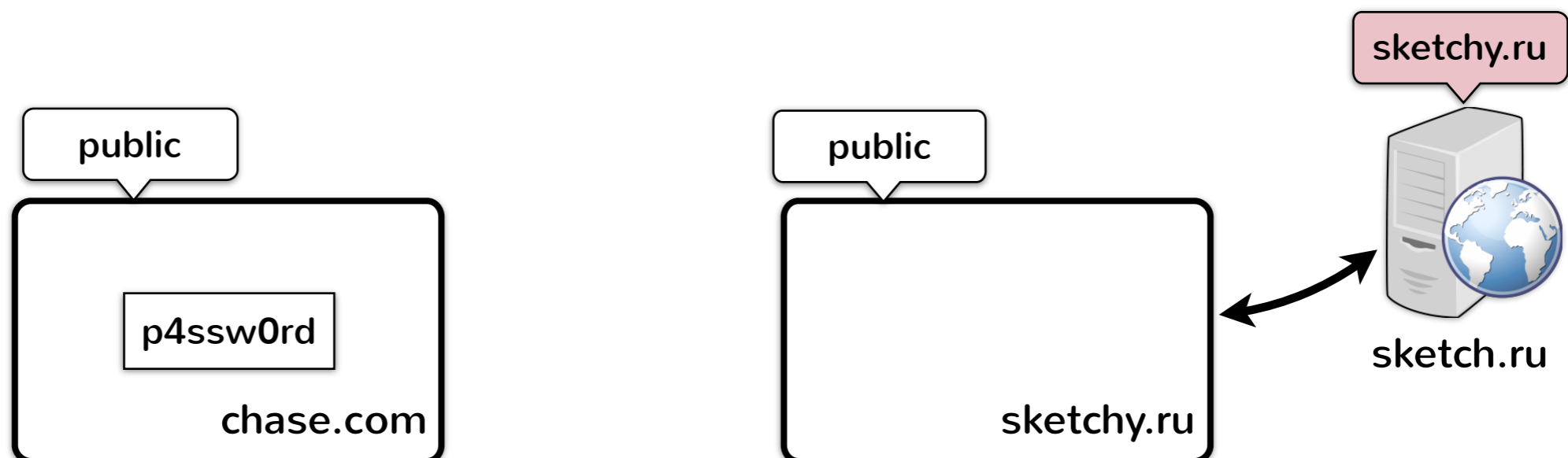
Labeled Communication

- Communication across browser contexts must respect label



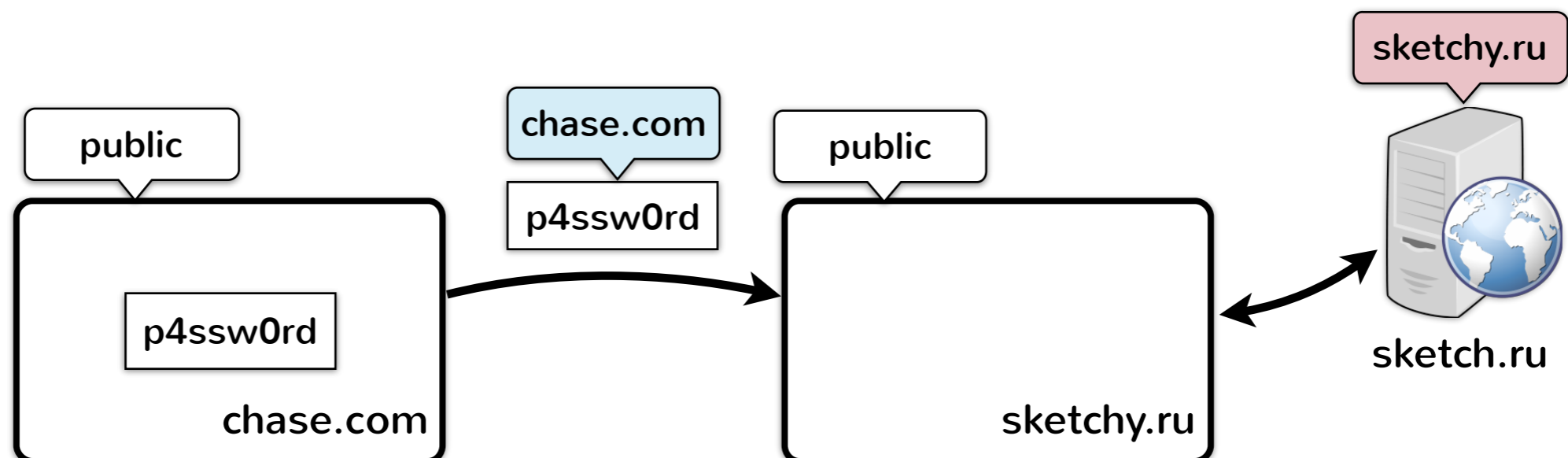
Adjusting labels to read data

- Contexts can adopt more restrictive label
 - I.e., add an origin to its label
 - Can then read data from that origin
 - Give up ability to write to contexts without it



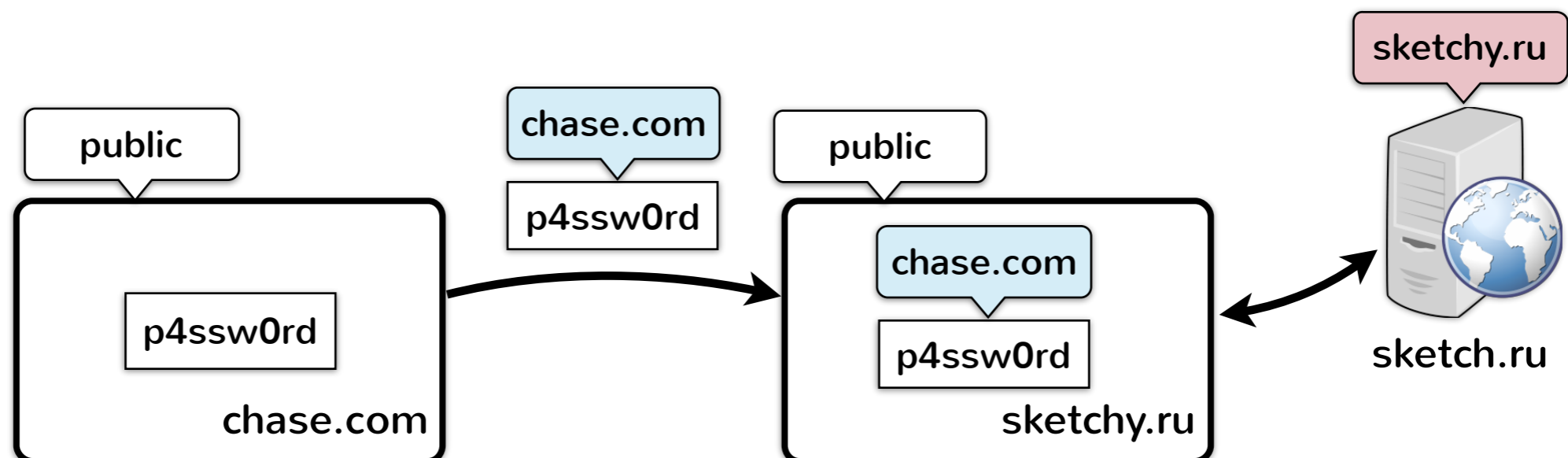
Adjusting labels to read data

- Contexts can adopt more restrictive label
 - I.e., add an origin to its label
 - Can then read data from that origin
 - Give up ability to write to contexts without it



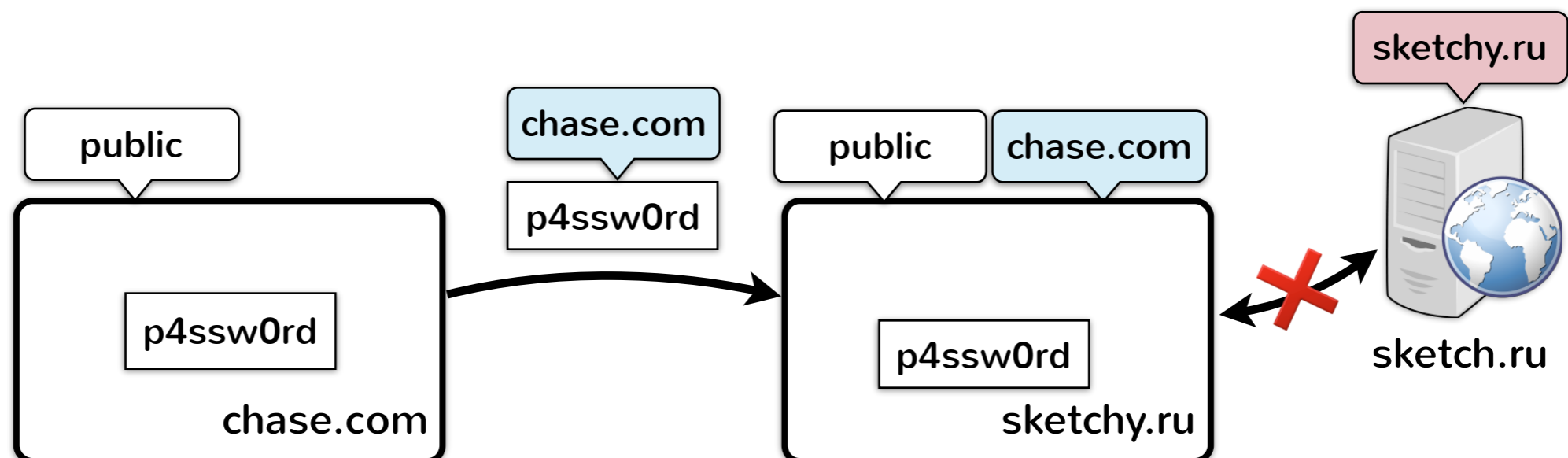
Adjusting labels to read data

- Contexts can adopt more restrictive label
 - I.e., add an origin to its label
 - Can then read data from that origin
 - Give up ability to write to contexts without it



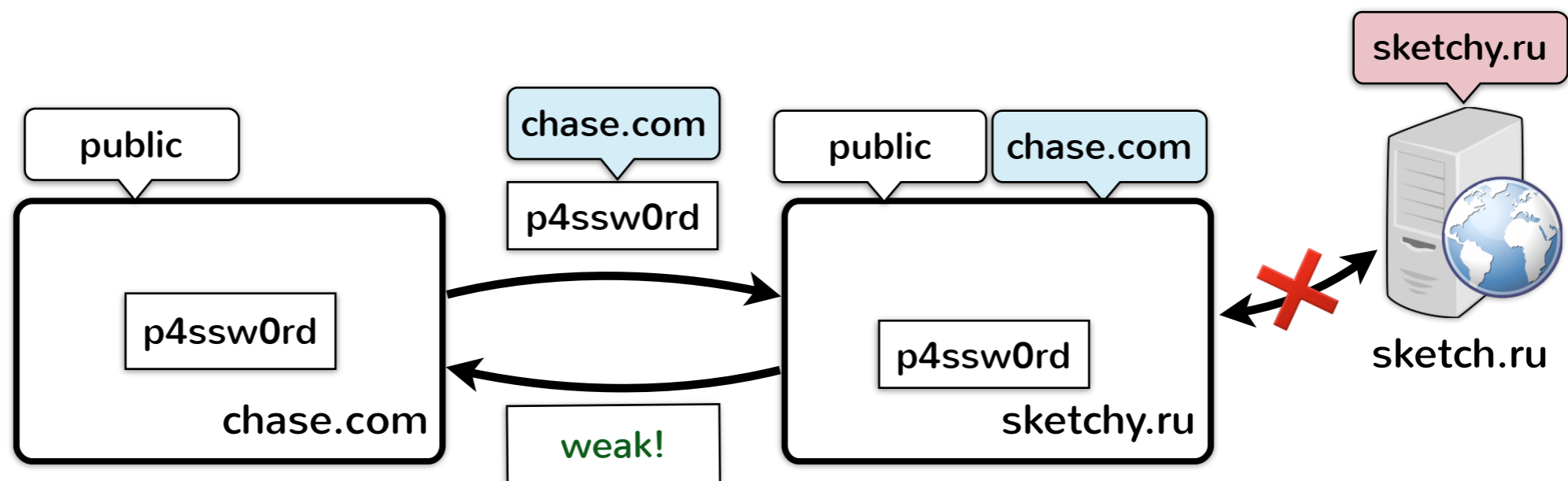
Adjusting labels to read data

- Contexts can adopt more restrictive label
 - I.e., add an origin to its label
 - Can then read data from that origin
 - Give up ability to write to contexts without it



Adjusting labels to read data

- Contexts can adopt more restrictive label
 - I.e., add an origin to its label
 - Can then read data from that origin
 - Give up ability to write to contexts without it



Summary: COWL design

Web was made for confinement

1. Origins are a natural way to specify labels
2. Leverage contexts as security boundaries
 - Mixed-granularity: label messages
3. Use origins to express privileges (see paper)

What can we do with this?

Example: client-side Mint

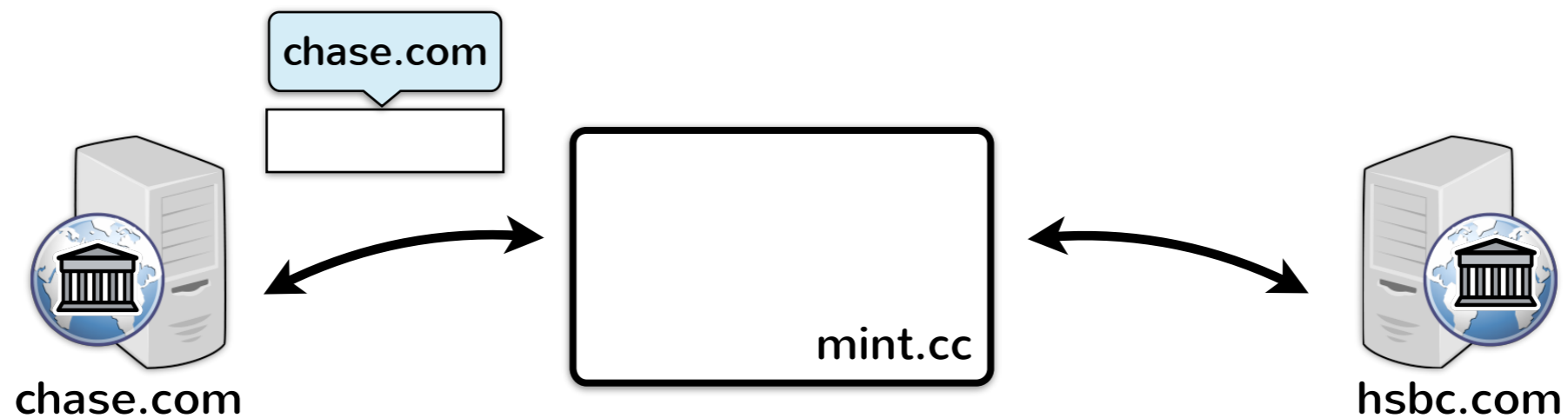
- Read-only client-side personal finance service



- Banks can make labeled statements available to Mint **⇒ Flexibility+Privacy!**

Example: client-side Mint

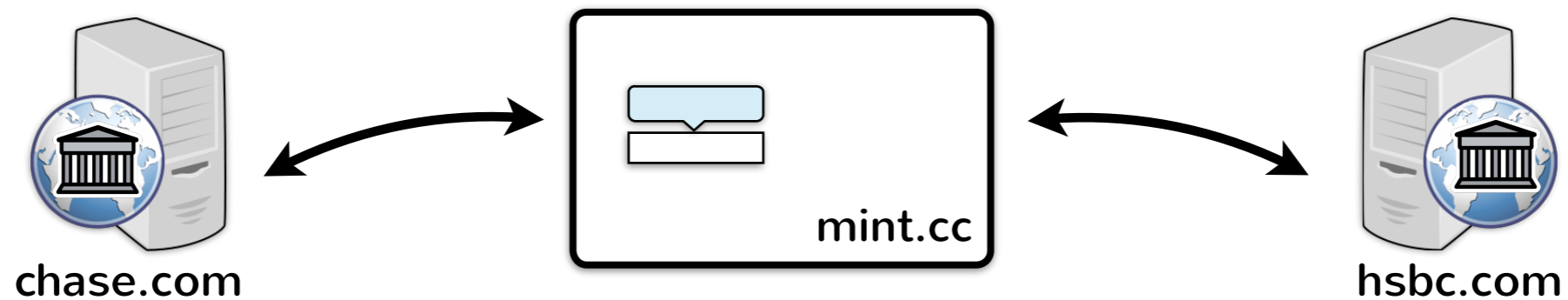
- Read-only client-side personal finance service



- Banks can make labeled statements available to Mint **Flexibility+Privacy!**

Example: client-side Mint

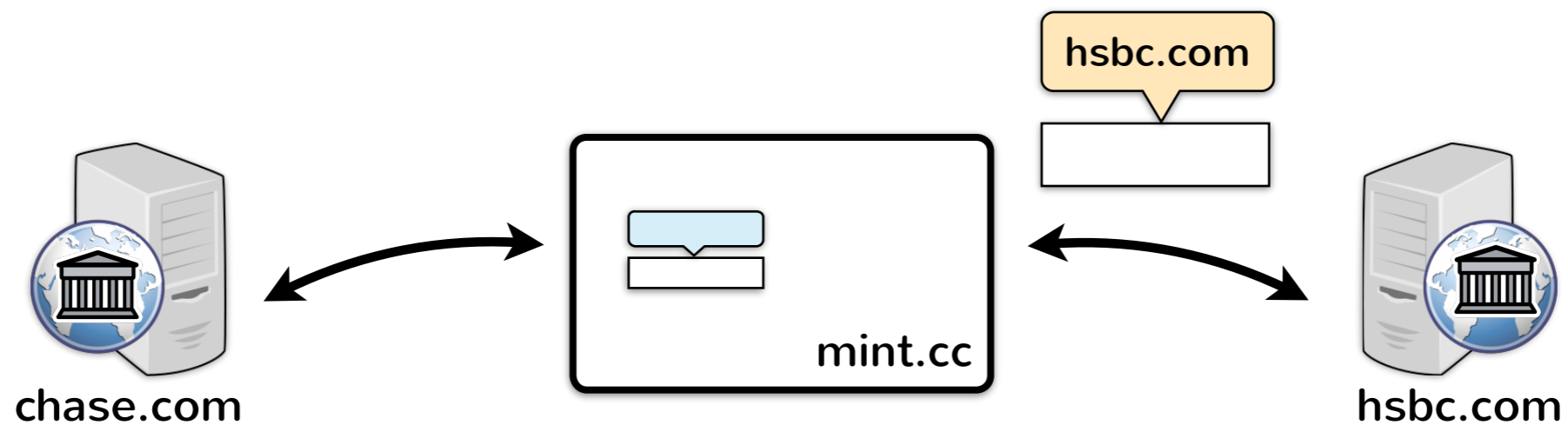
- Read-only client-side personal finance service



- Banks can make labeled statements available to Mint **⇒ Flexibility+Privacy!**

Example: client-side Mint

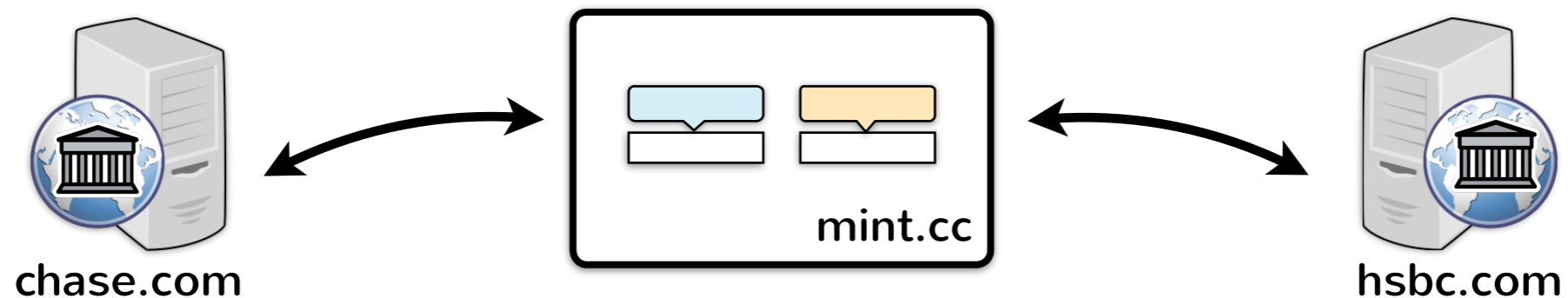
- Read-only client-side personal finance service



- Banks can make labeled statements available to Mint **Flexibility+Privacy!**

Example: client-side Mint

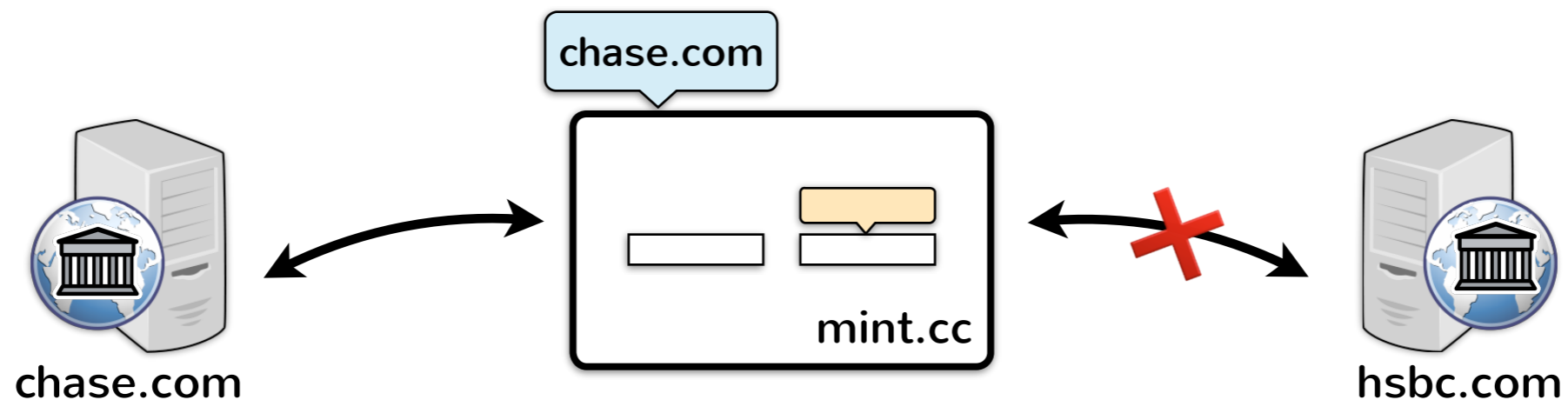
- Read-only client-side personal finance service



- Banks can make labeled statements available to Mint **Flexibility+Privacy!**

Example: client-side Mint

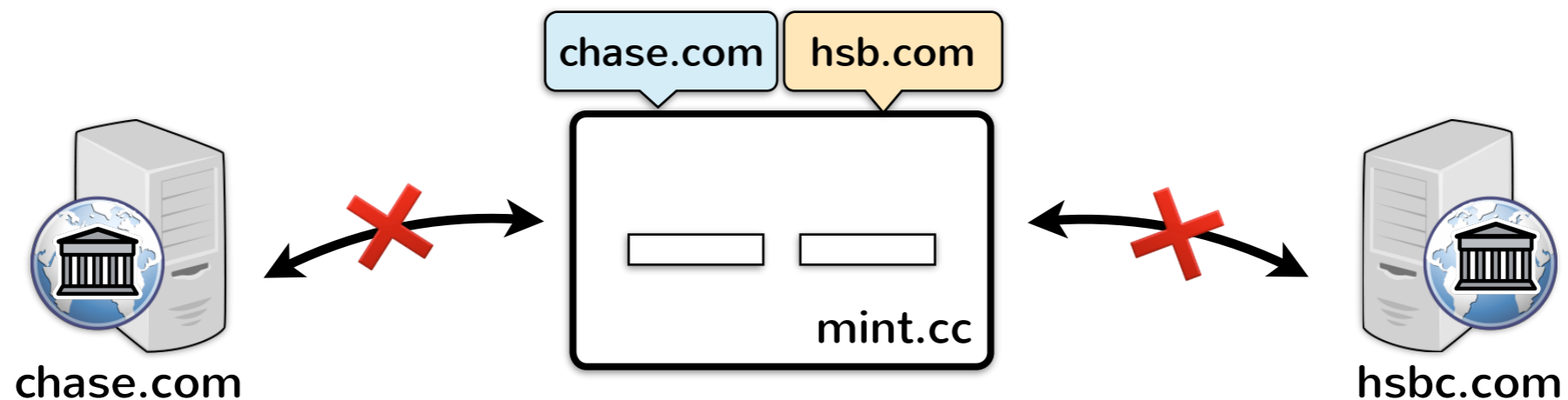
- Read-only client-side personal finance service



- Banks can make labeled statements available to Mint \Rightarrow **Flexibility+Privacy!**

Example: client-side Mint

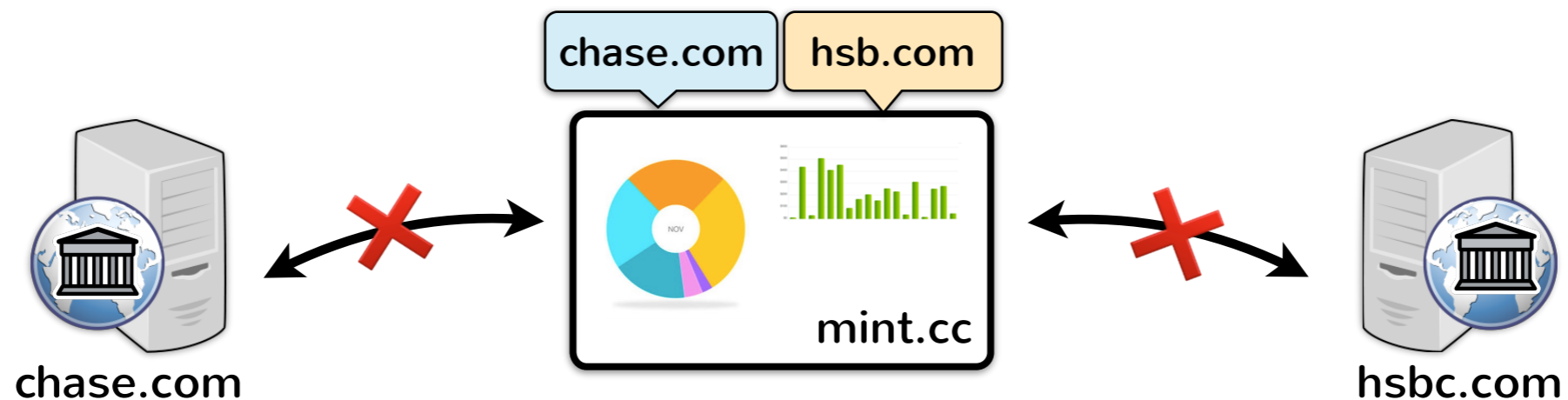
- Read-only client-side personal finance service



- Banks can make labeled statements available to Mint **Flexibility+Privacy!**

Example: client-side Mint

- Read-only client-side personal finance service



- Banks can make labeled statements available to Mint **Flexibility+Privacy!**

We built it...

Implementations

- **DOM-level API for both Firefox and Chromium**
 - No changes to JavaScript engines
 - Maintain existing communication APIs
 - For each page COWL only enabled on first use of API
- **Gecko and Blink: roughly 4K lines of C++ each**

Evaluation: Performance

- Overhead of securing a mashup service?
- Overhead of compartmentalization?
- Will adding COWL slow the existing Web?

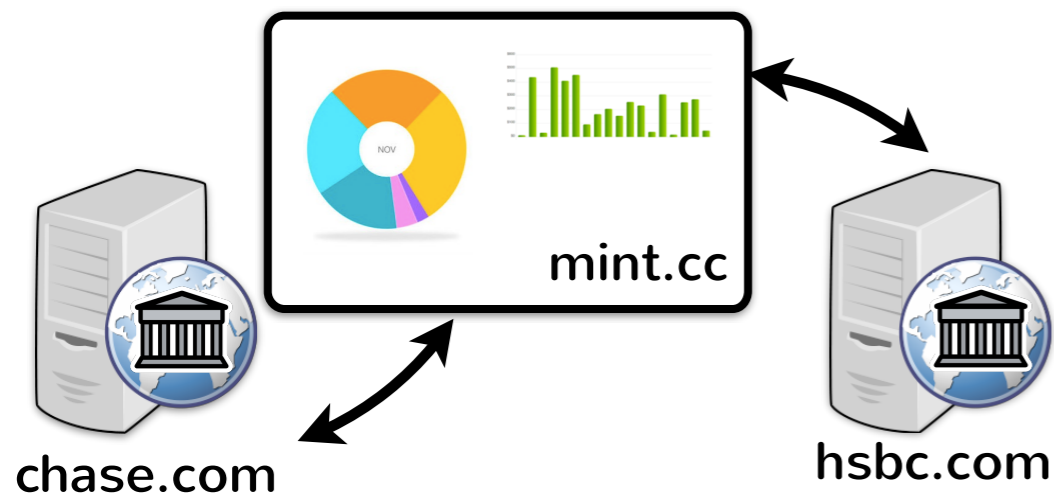
Evaluation: Performance

Worst-case (loopback, trivial app code)
end-to-end page load: roughly 16% [16ms]

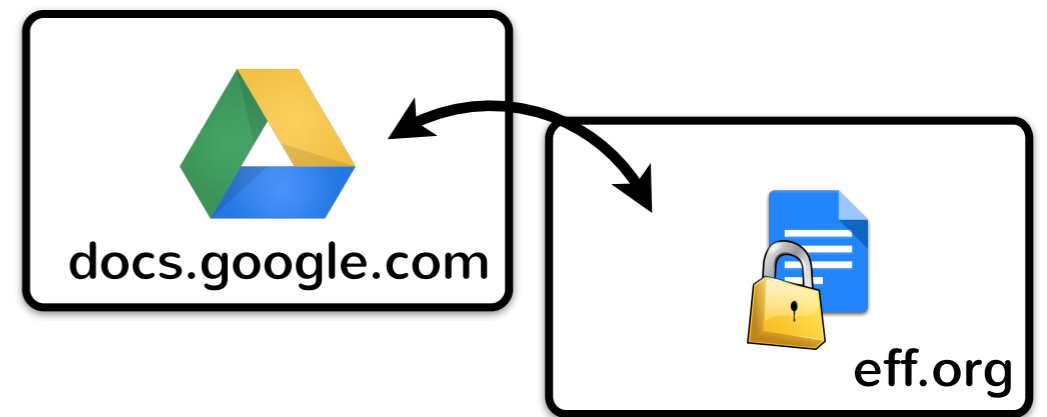
For real apps: **relative overhead is small!**

Evaluation: Applicability

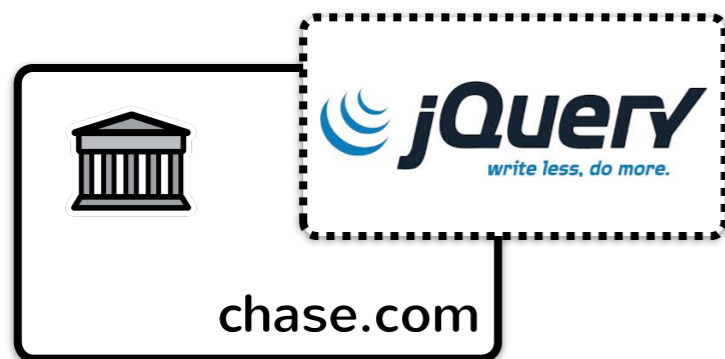
Third-party mashups



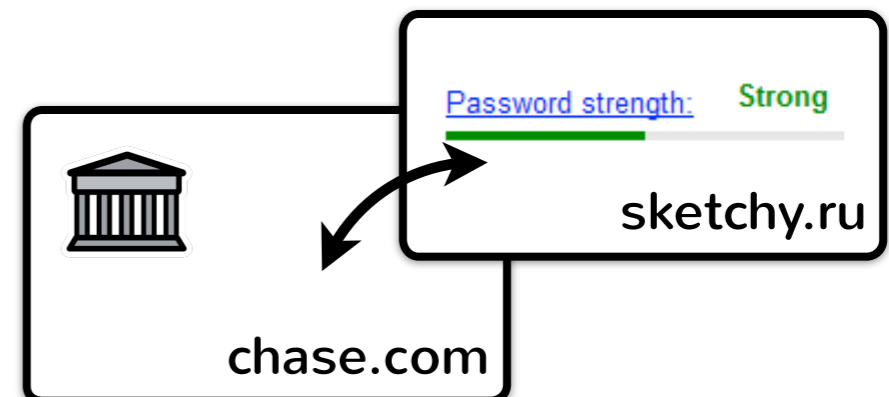
Mutually distrusting services



Tightly-coupled libraries



Libraries with narrow APIs



Deployability

- **High degree of backward compatibility**
 - Does not affect pages that do not use COWL API
- **Reuse existing concepts (origins, contexts)**
 - Expect it to be friendly to developers
- **Implementations possible for major browsers**
 - Changes don't touch JavaScript engine

Limitations & future work

- **Covert channels**
 - Malicious code may still covertly leak data
 - COWL enforces MAC in addition to existing DAC
- **Compartmentalization**
 - Cannot just label and run existing apps
 - Compartmentalizing applications requires thought

Related work

- **Coarse-grained confinement: BFlow**
 - Mainly concerned with untrusted code
 - COWL also handles the mutually distrusting case
- **Fine-grained confinement: JSFlow**
 - Better fit for tightly-coupled libraries
 - New semantics, 100x slowdown

Conclusion

Today: give up privacy for flexibility

- Modern web apps need to compute on sensitive data
- DAC is crucial, but insufficient!

COWL: confinement for client-side code

- Naturally extends the existing web model
- Achieves both flexibility and privacy without slowdown

Thanks!

<http://cow1.ws>