Confinement with Origin Web Labels (COWL)

Deian Stefan

STANFORD UNIVERSITY

http://cowl.ws

Today

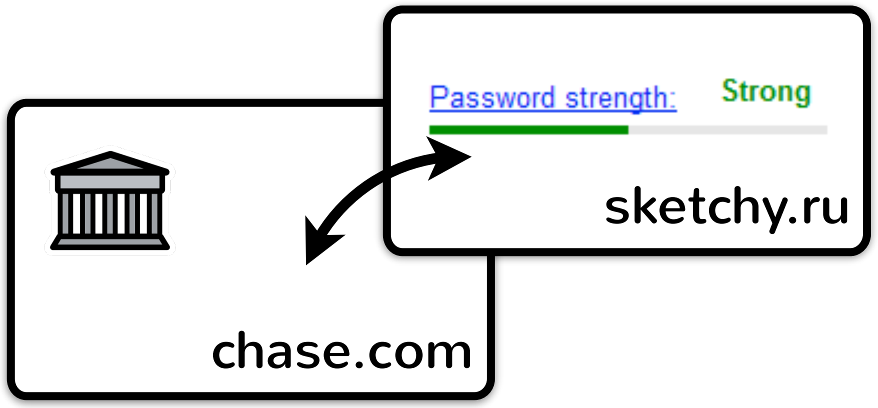
$$\left.\begin{array}{r}\text{SOP}\\ \text{CSP}\\ \text{CORS}\end{array}\right\} \begin{array}{l}\textbf{D}\text{iscretionary}\\ \textbf{A}\text{ccess}\\ \textbf{C}\text{ontrol}\end{array}$$

➠ Crucial for securing the Web!

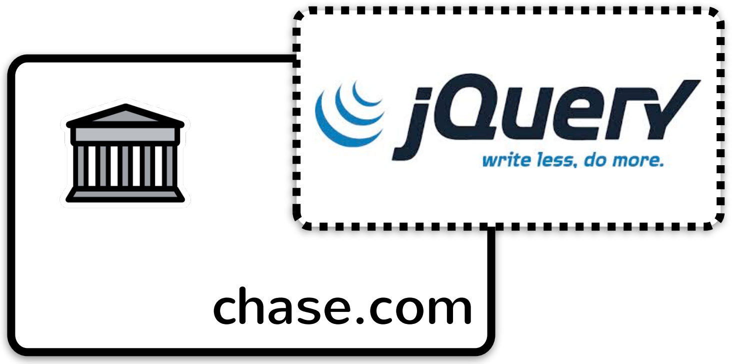
But fall short in a few cases...

Where DAC falls short...

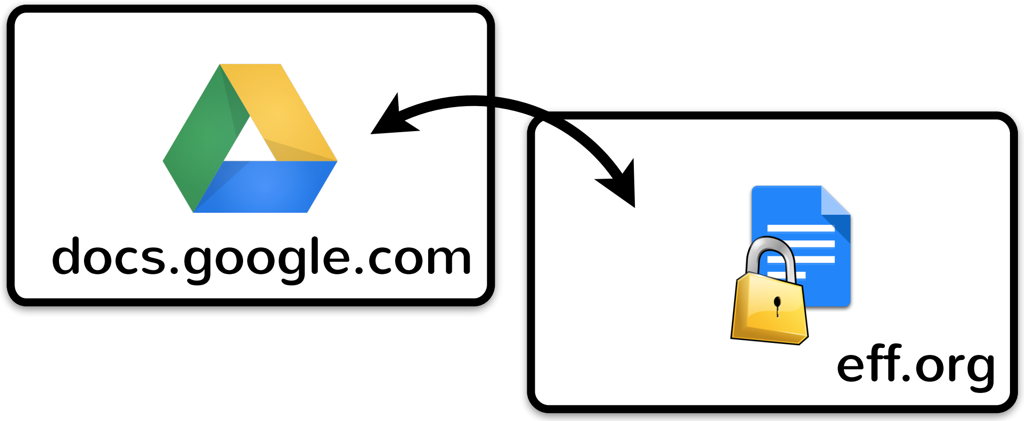
Libraries with narrow APIs



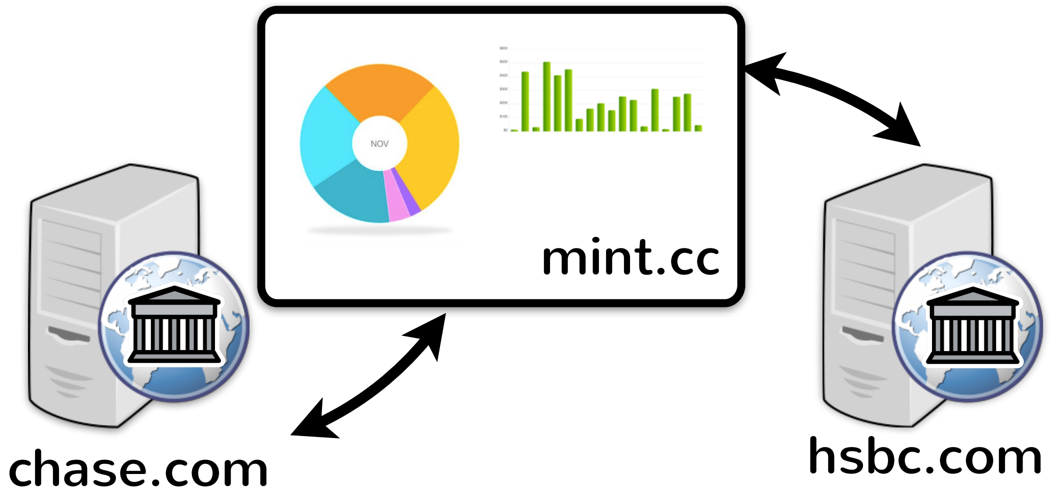
Tightly-coupled libraries



Mutually distrusting services



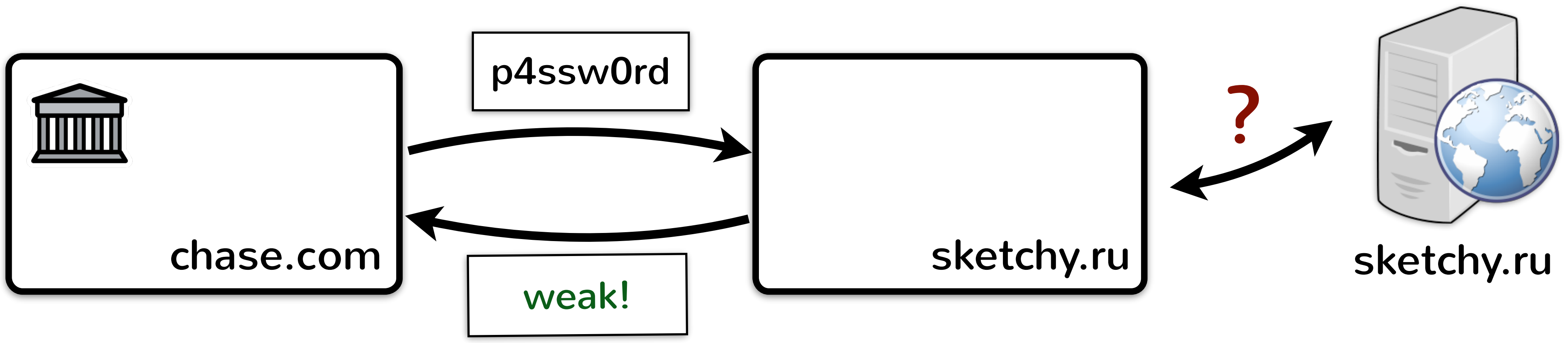
Third-party mashups



How does DAC fall short?

Forces choice between functionality and privacy

- E.g., password strength checker library

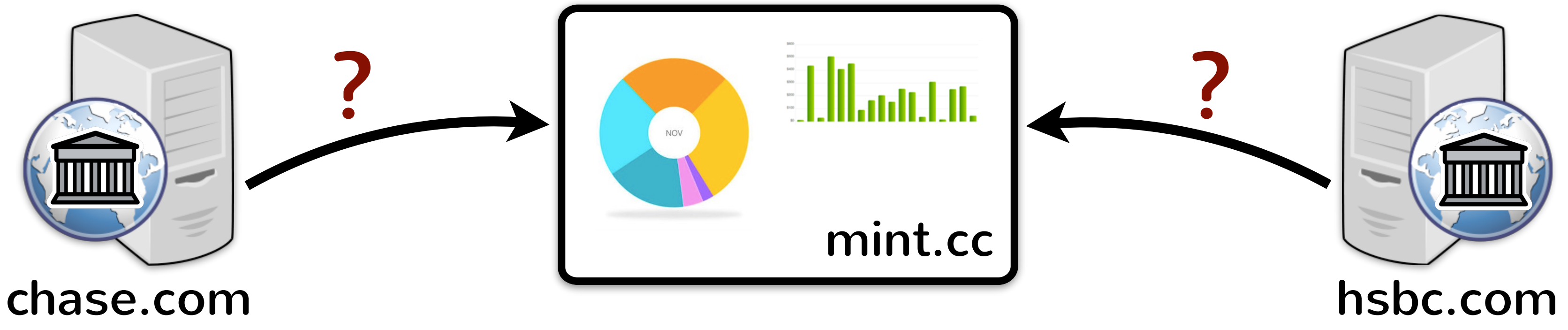


- **Privacy:** use CSP+sandbox to disallow communication
- **Functionality:** allow checker to fetch common pass.

How does DAC fall short?

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)

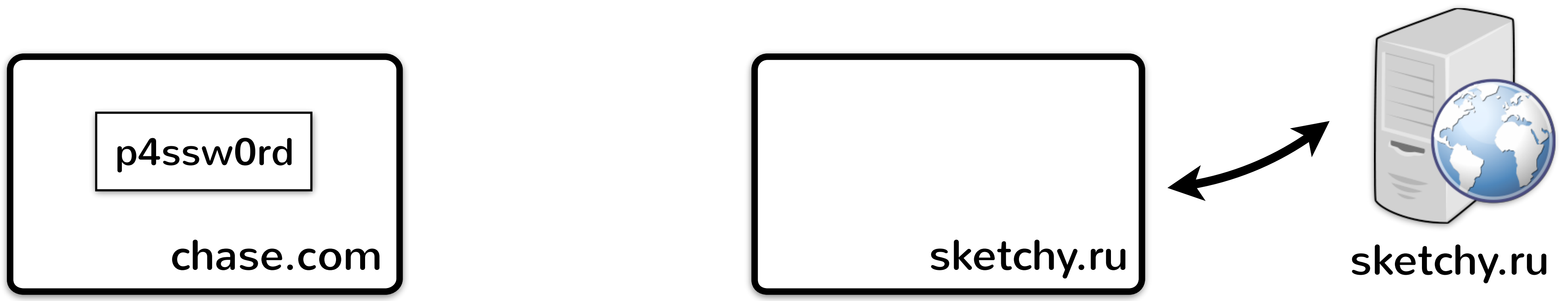
Why does DAC fall short?

- **Fundamentally**
  - Apps rely on and use third-party code
  - This code computes on sensitive data
- **DAC restricts who can access data**
  - Not what code can do with the data once granted access!

Confinement (at a glance)

Idea: impose restrictions on how code uses data

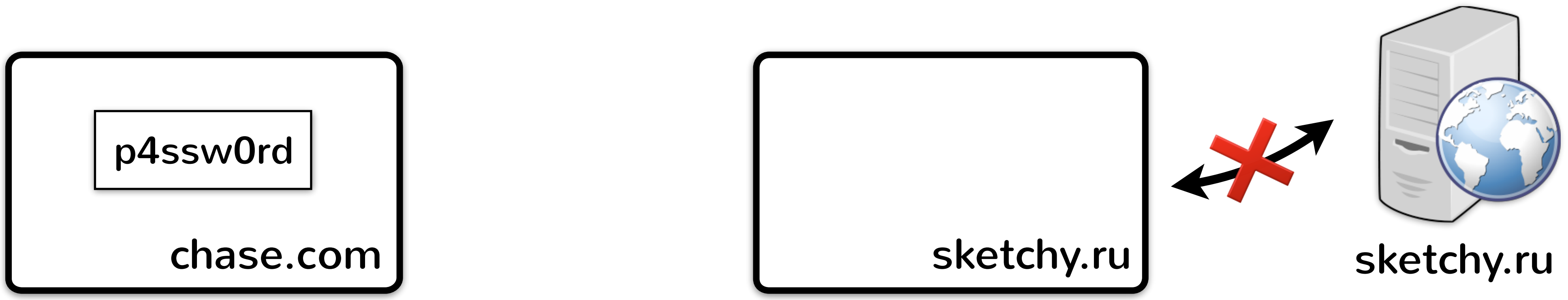
- E.g., it is safe to fetch list of common password before looking at password, but once password is inspected
  - ➠ restrict communication!



Confinement (at a glance)

Idea: impose restrictions on how code uses data

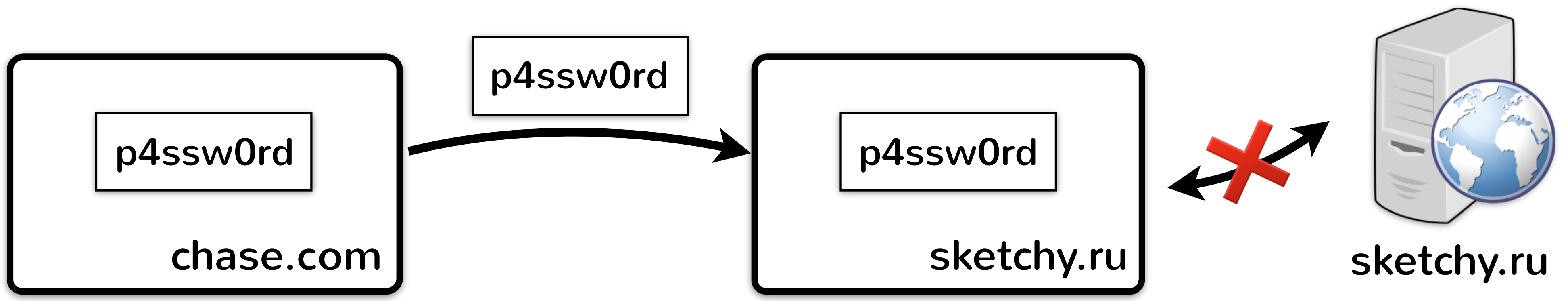
- E.g., it is safe to fetch list of common password before looking at password, but once password is inspected
  - ➠ restrict communication!



Confinement (at a glance)

Idea: impose restrictions on how code uses data

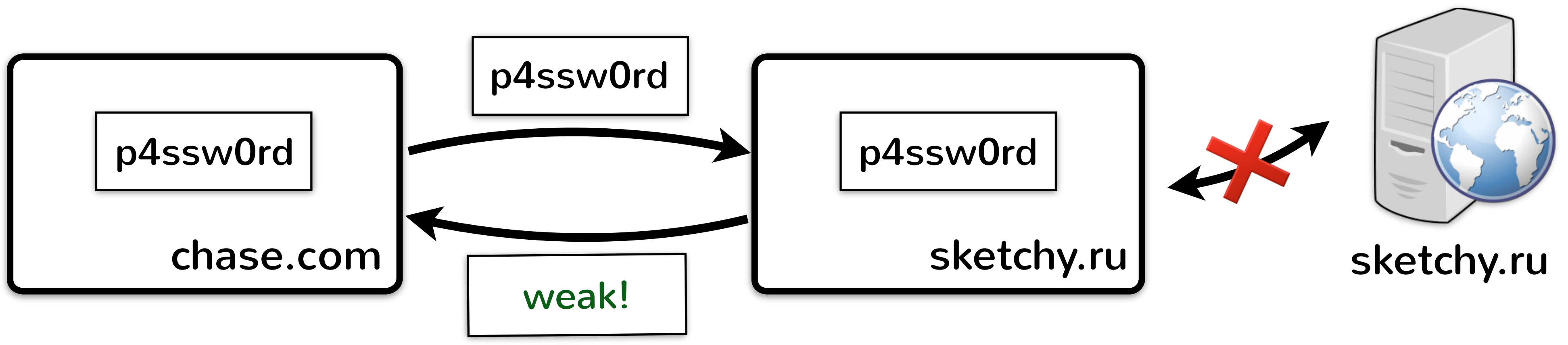
- E.g., it is safe to fetch list of common password before looking at password, but once password is inspected
  - ➠ restrict communication!



Confinement (at a glance)

Idea: impose restrictions on how code uses data

- E.g., it is safe to fetch list of common password before looking at password, but once password is inspected
  ➠ restrict communication!



COWL design

Extend browser with

1. Labels: policies specified in terms of origins
    - Way for developers to express security concerns
2. Label tracking/enforcement
3. Privileges: extend SOP's notion of trust
    - Avoid being confined for reading own data

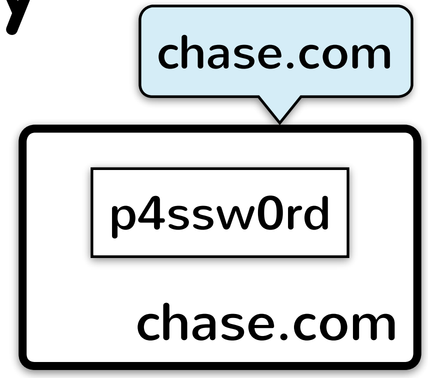
Labels

Label specifies, in terms of origin(s), who cares about the data

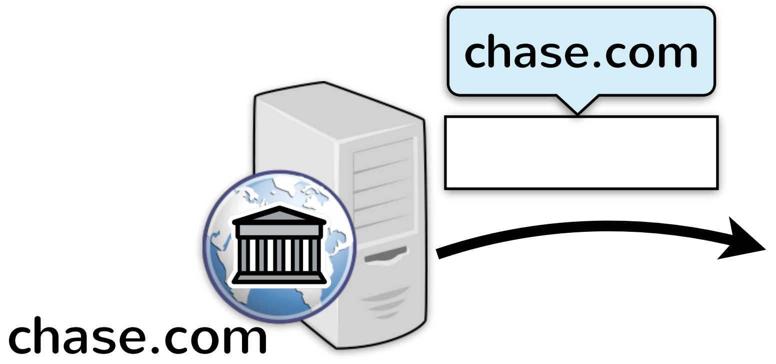
- E.g., data sensitive to Chase: `Label("chase.com")`
- E.g., data sensitive to Alice on Twitter [like sub-origin]: `Label("twitter.com").or("@alice")`
- E.g., data sensitive to both Chase and HSBC: `Label("chase.com").and("hsbc.com")`

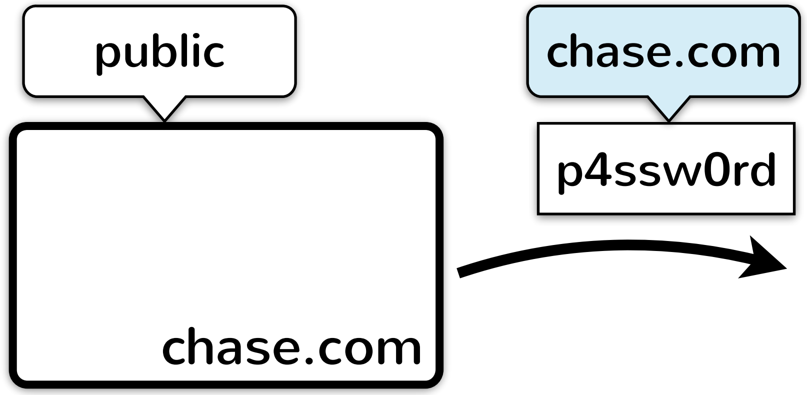
Label tracking

- **COWL tracks labels at context granularity**
  - Pages, iframes, workers, and light-weight workers (new LWorker API)



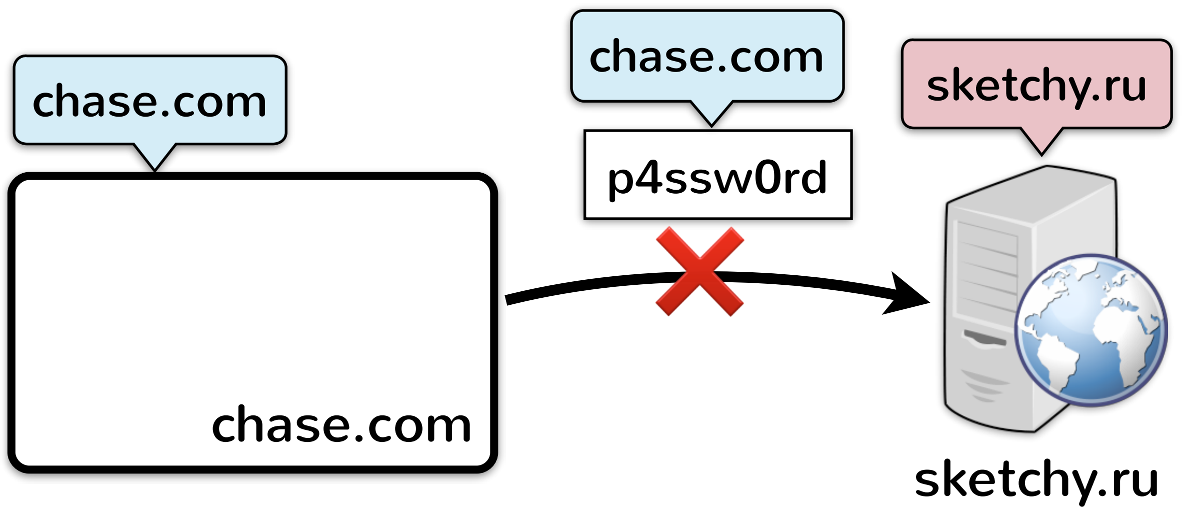
- **Messages can be labeled differently from context**
  - Both servers & JavaScript can label messages





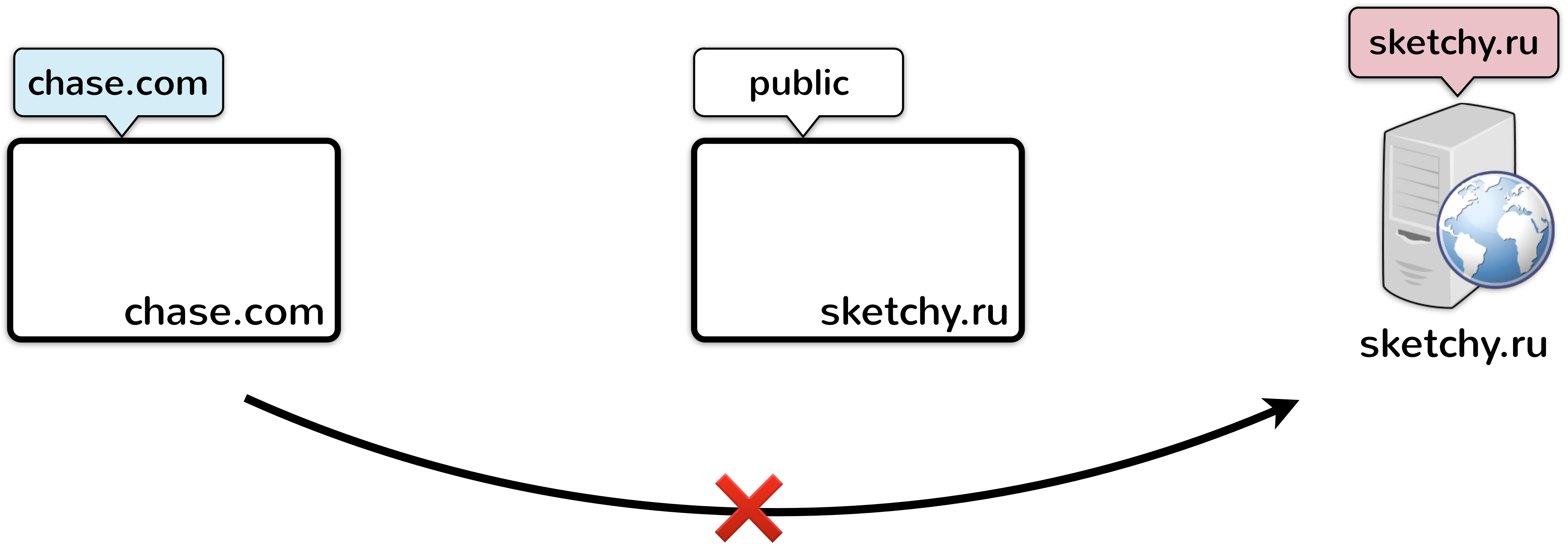
Label enforcement

Browser-server communication must respect labels!



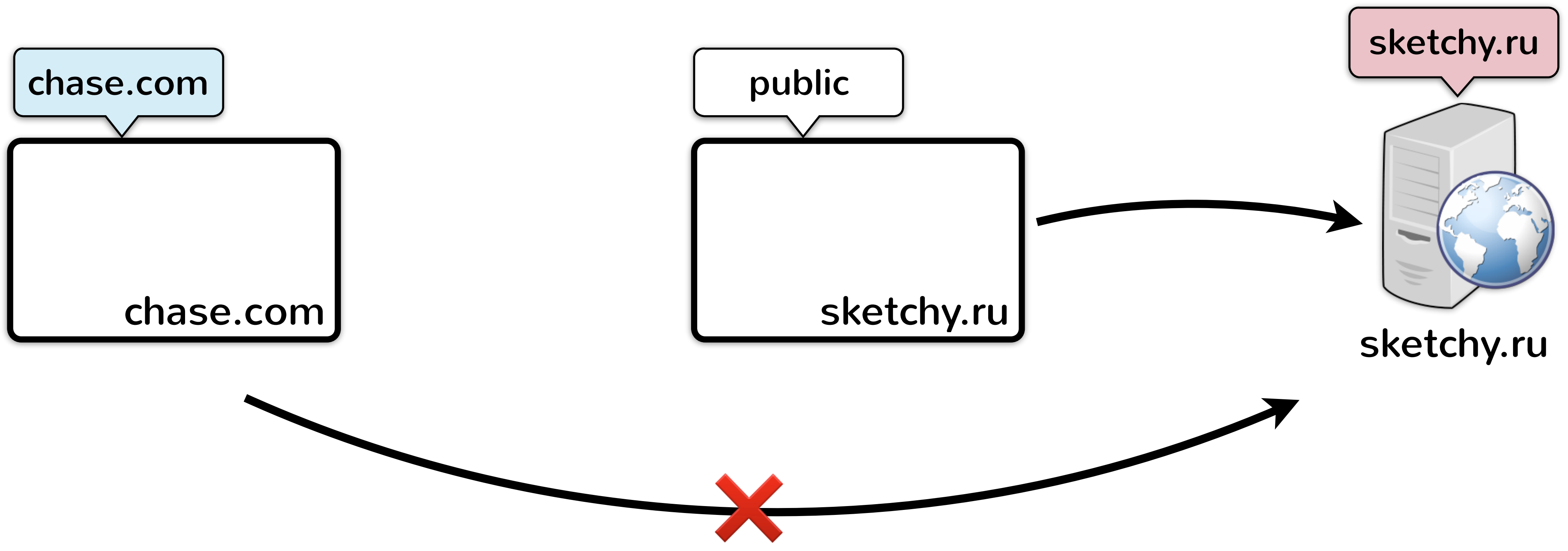
Label enforcement

Cross-context communication must respect labels!



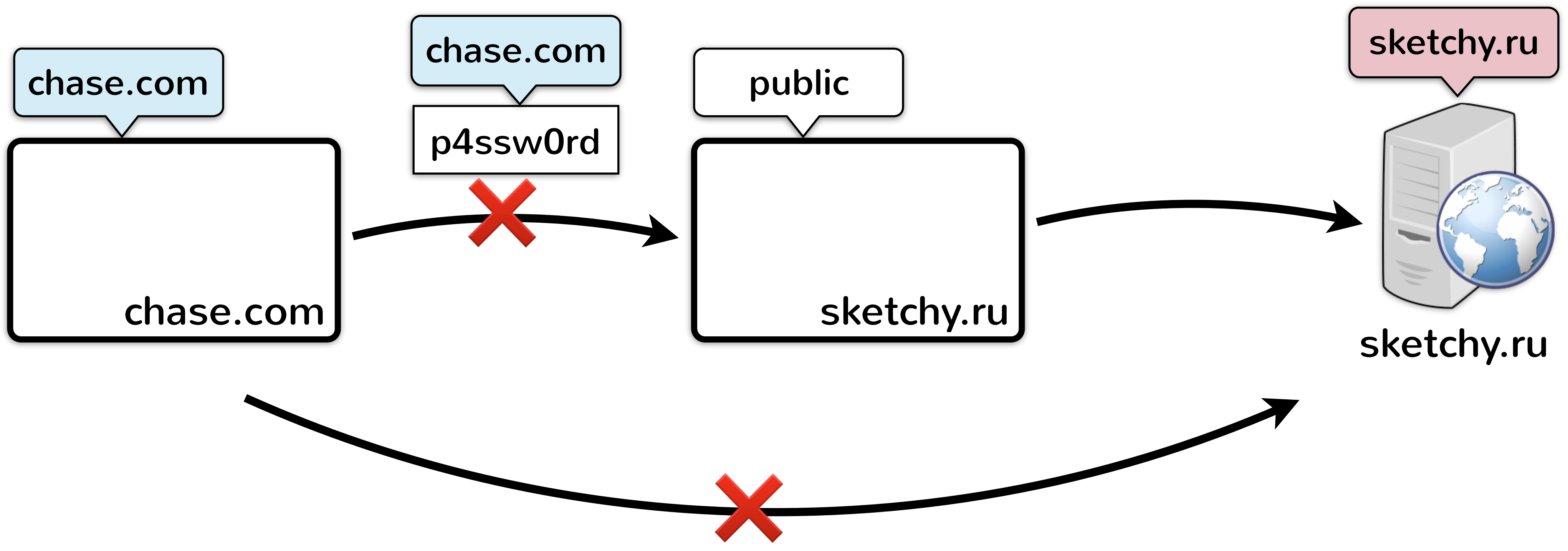
Label enforcement

Cross-context communication must respect labels!



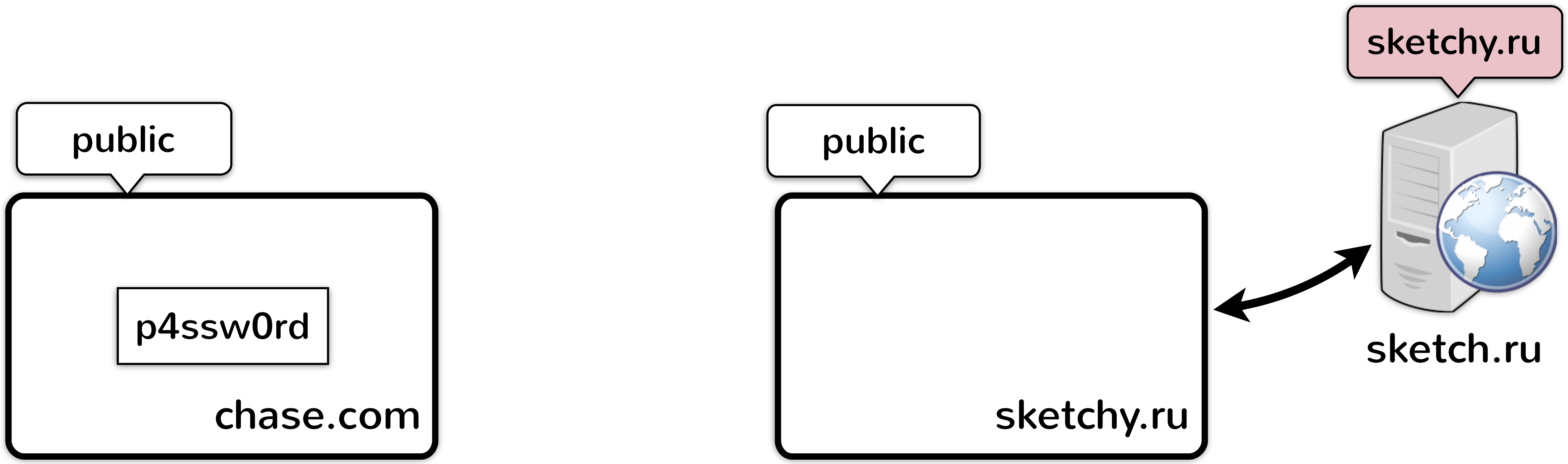
Label enforcement

Cross-context communication must respect labels!



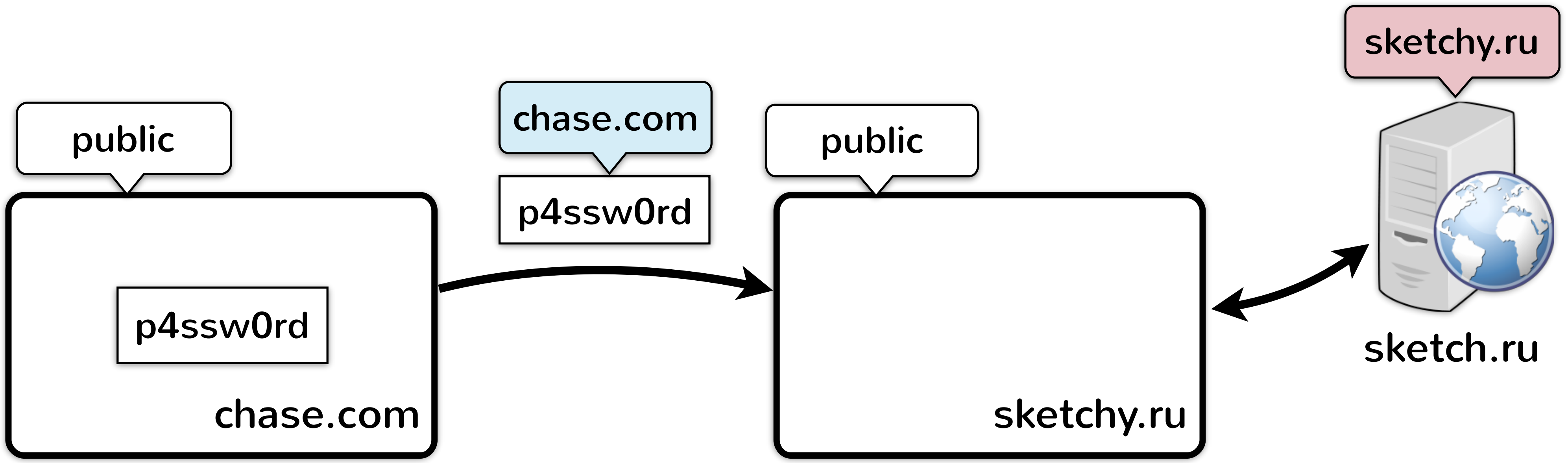
Adjusting labels to read data

- 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



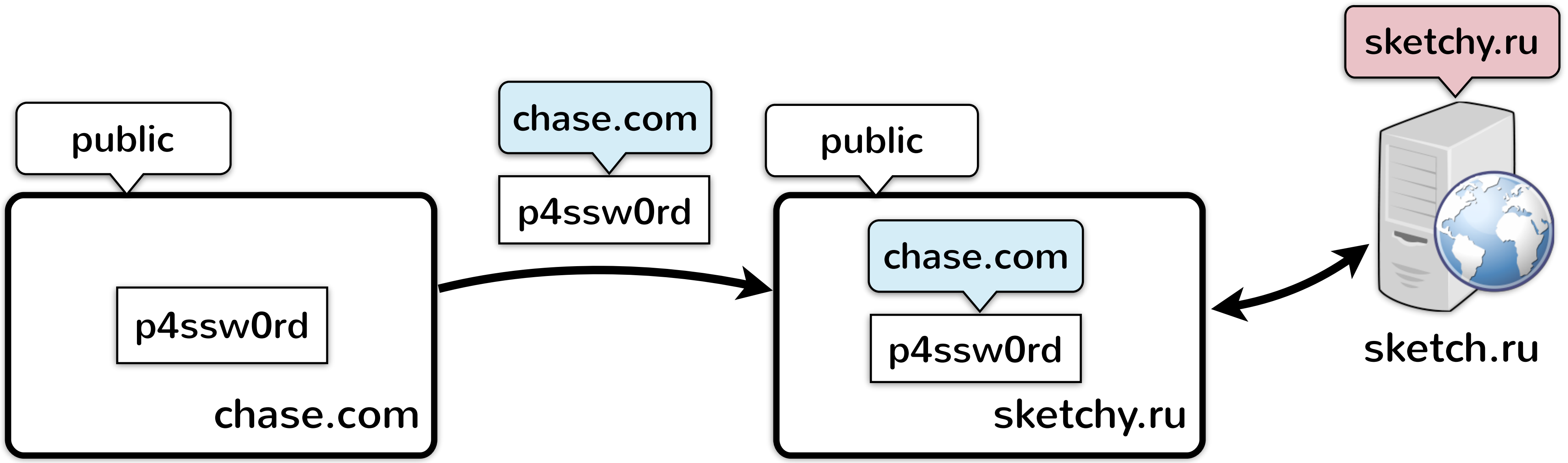
Adjusting labels to read data

- 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



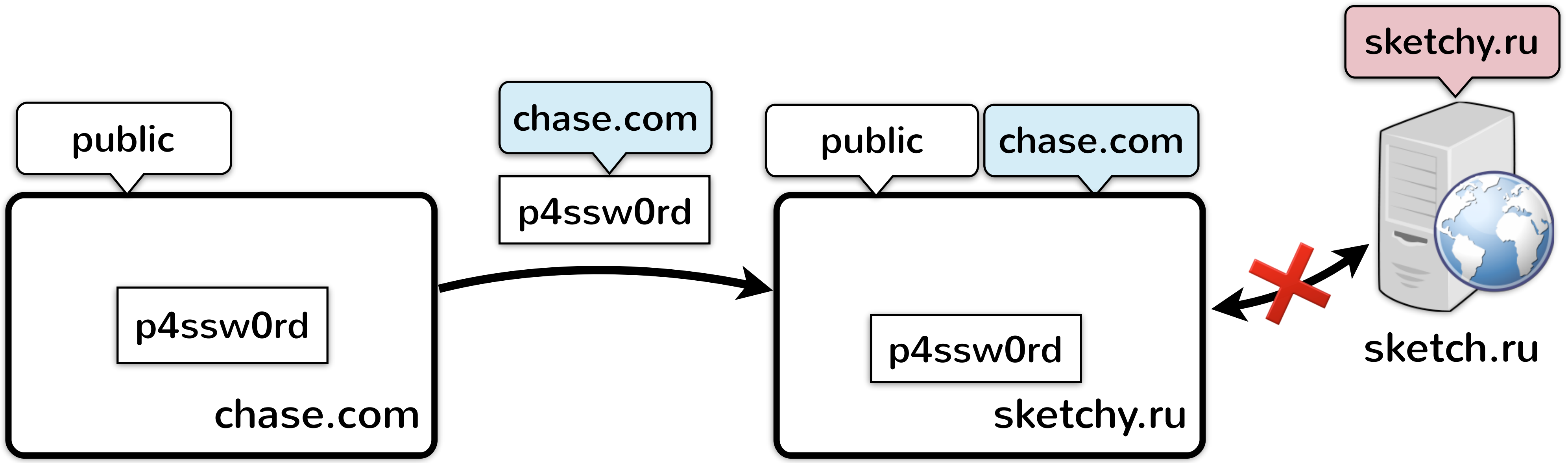
Adjusting labels to read data

- 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



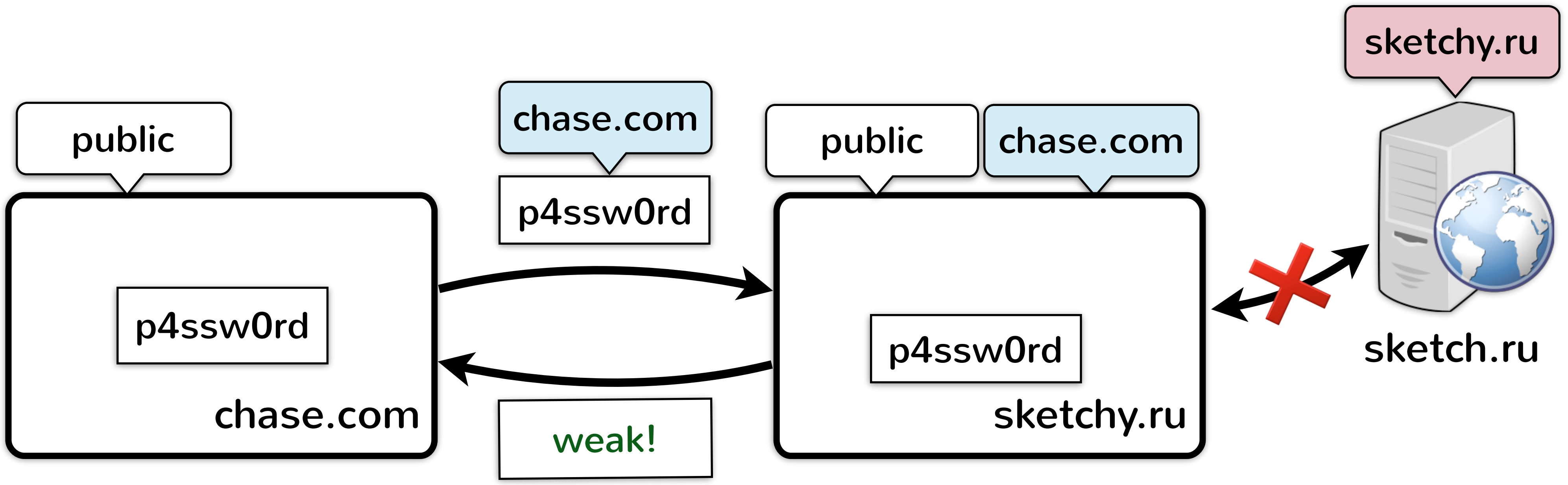
Adjusting labels to read data

- 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



Privileges

- **Page dictates how data of its origin gets disseminated**
  - As in SOP: page is trusted with its own data
- **COWL makes this explicit with privileges**
  - Context has unforgeable Privilege object
  - No confinement by labels corresponding to privileges
  - Unlike SOP: privileges can be dropped & delegated

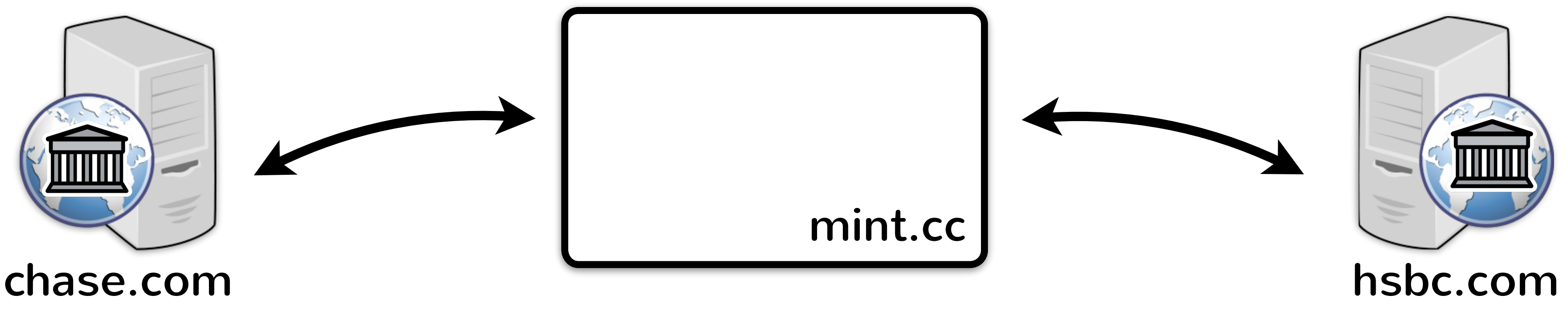
Summary: COWL design

- **Origins are a natural way to specify policy**
  - Conjunction specifies concern of multiple origins
  - Disjunctions (or) specifies “sub-origin concerns”
- **Leverage contexts as security boundaries**
  - Impose restrictions on code by labeling messages
  - Use LWorkers to confine code (vs. <script>’s)

What can we do with this?

Example: client-side Mint

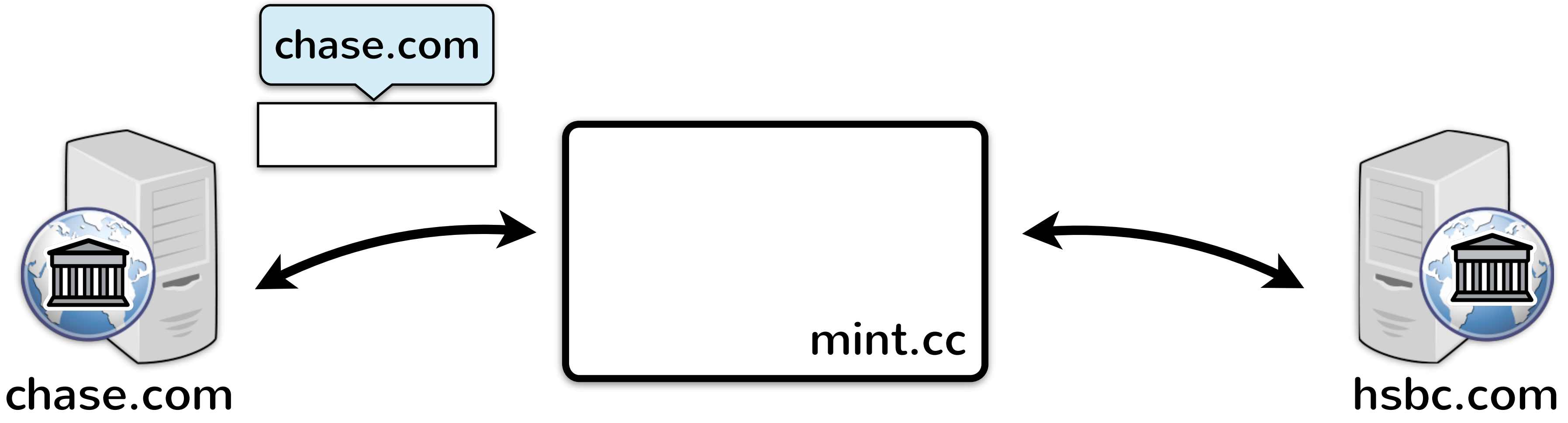
- Read-only client-side personal finance service



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

Example: client-side Mint

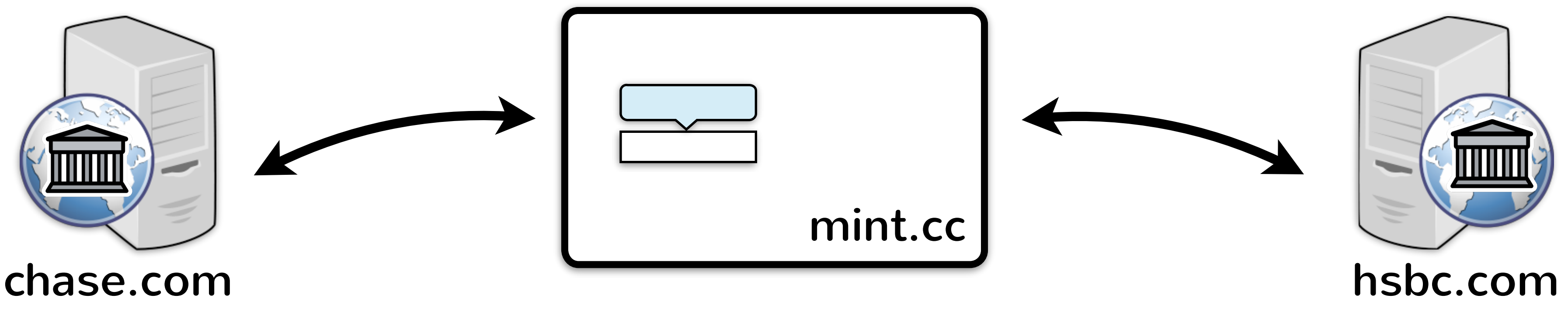
- Read-only client-side personal finance service



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

Example: client-side Mint

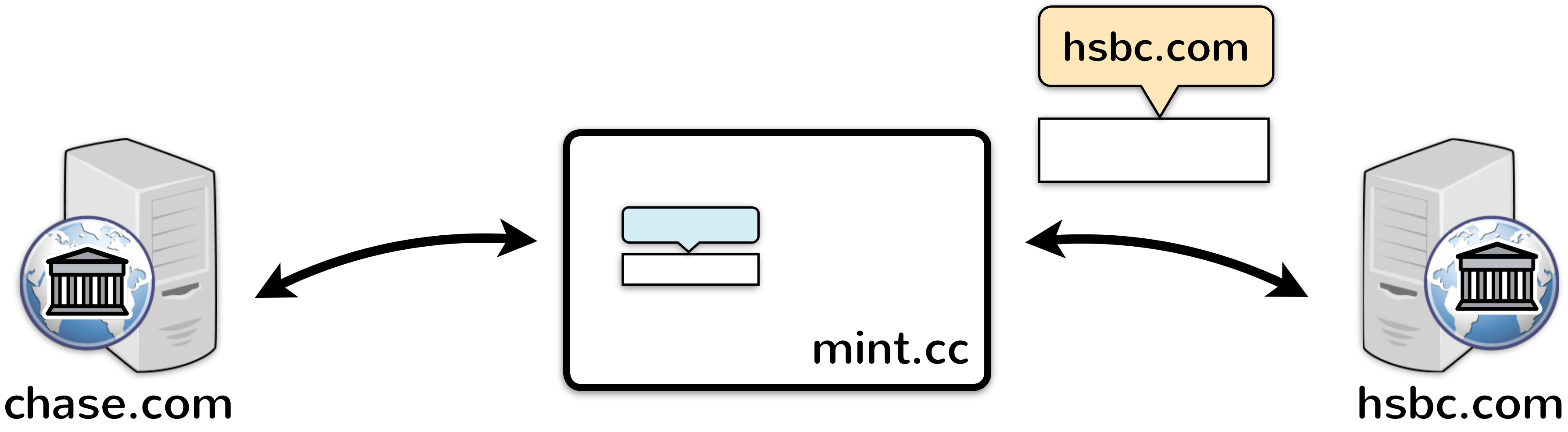
- Read-only client-side personal finance service



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

Example: client-side Mint

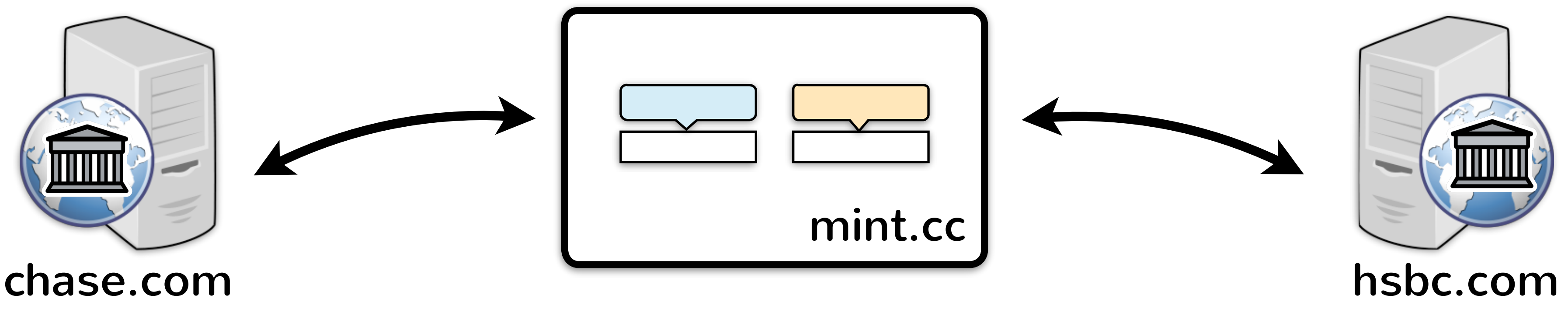
- Read-only client-side personal finance service



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

Example: client-side Mint

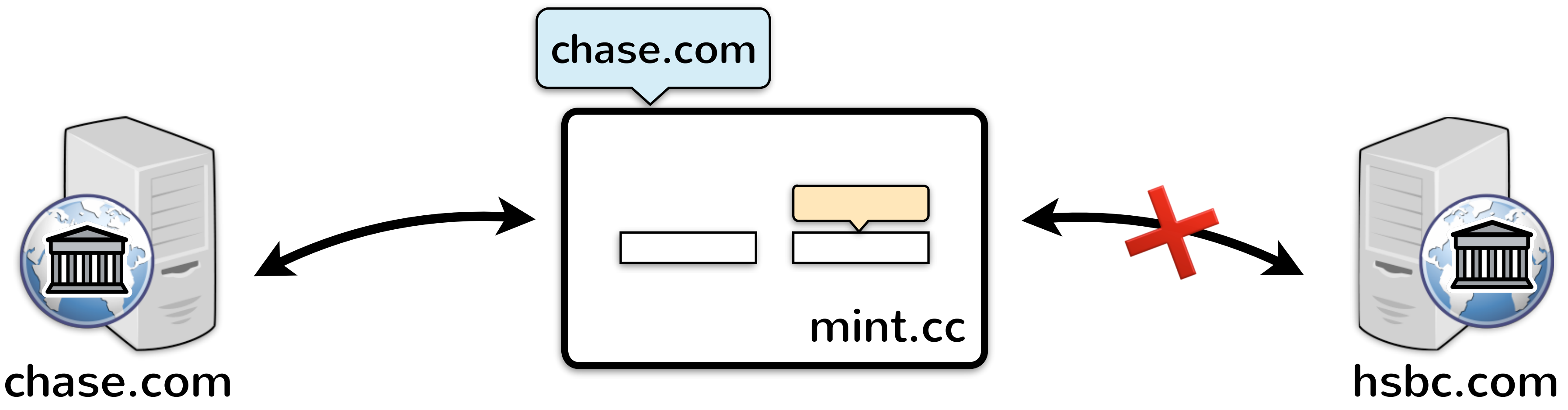
- Read-only client-side personal finance service



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

Example: client-side Mint

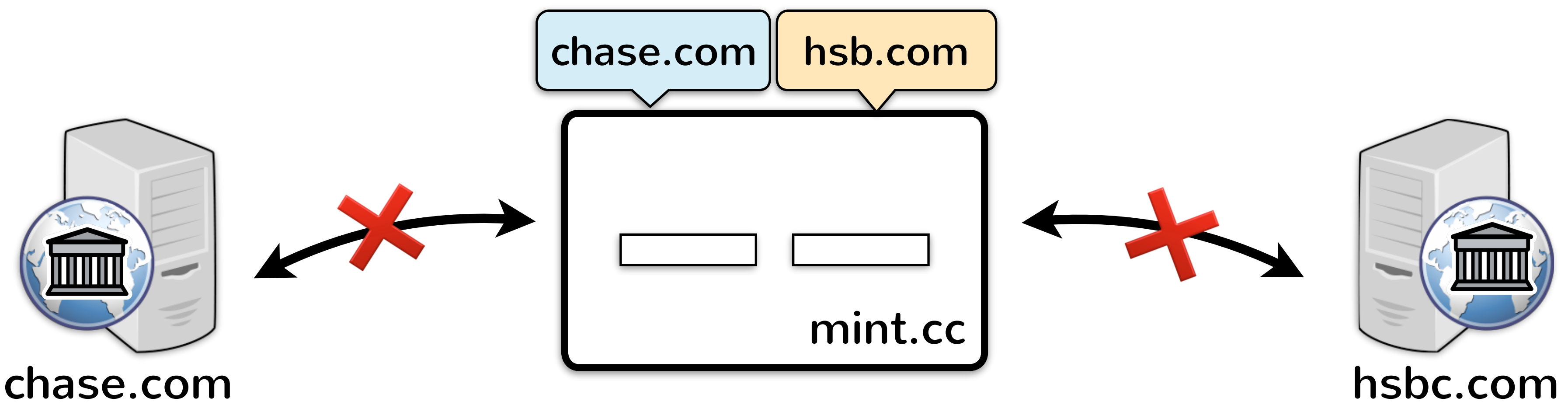
- Read-only client-side personal finance service



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

Example: client-side Mint

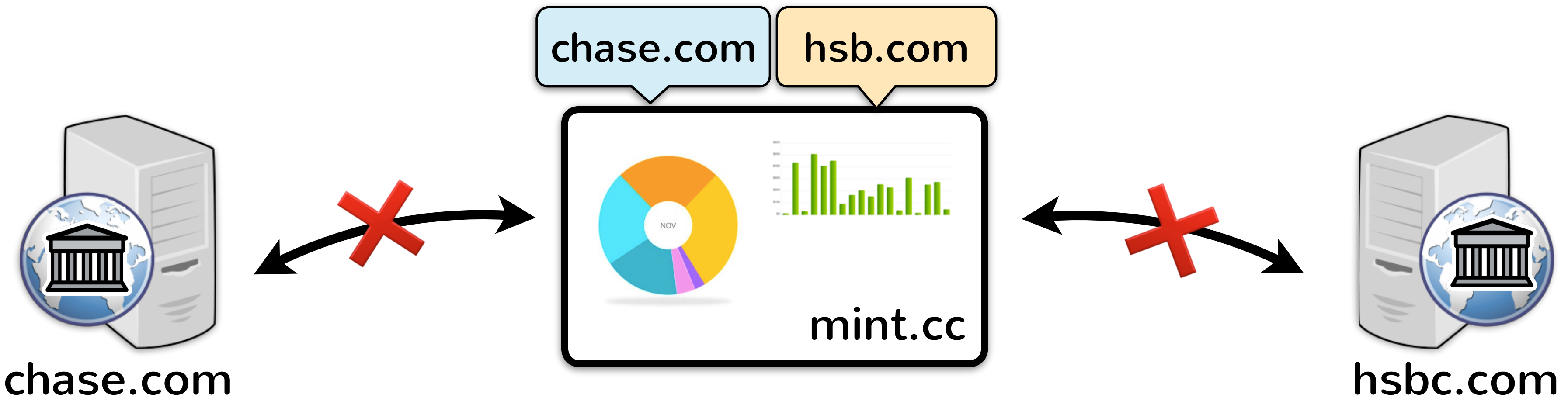
- Read-only client-side personal finance service



- 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!

Demo: third-party library

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**
- **Current status: porting to latest FF & Chromium**

Label enforcement

- **Piggy-backing on CSP+sandbox**
  - CSP effectively allows us to control where context can disseminate data
  - We adjust underlying context CSP according to label of context
- **Cross-context communication**
  - Gecko: new compartment wrappers
  - Blink: modified DOM bindings

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!

Deployability

- **High degree of backward compatibility**
  - Does not affect pages that do not use COWL API, functionality or performance-wise
- **Reuse existing concepts (origins, contexts)**
  - Expect it to be friendly to developers

Intersection with other proposals

- Issue 69: Overt channel control in CSP
- Scriptable CSP proposal
- Sub-origins proposal
    - Key difference: labels are explicit and visible
- Sandboxed Cross-Origin Workers
- LWorkers may be useful for bookmarklets?

Future direction

- **LWorkers can access parent DOM if given privilege**
    - ➤ Effectively: reverse sandbox
    - ➤ Next step: tie in with shadow DOM to allow untrusted code in LWorker to modify part of page

Thanks!

`http://cowl.ws`