Hails: Protecting Data Privacy in Untrusted Web Applications

Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei,
John Mitchell, David Mazières, and Alejandro Russo

github:develop

Web platforms are **great**!

They allow third-party developers to build apps that use our personal data.

Google Developers

foursquare DEVELOPERS

PCMAG.COM

Path Uploads Your Entire iPhone Contact List By Default

THE WALL STREET JOURNAL.

Home | World | Europe | U.K. | U.S. | Business | Markets | Market Data

WHAT THEY KNOW | October 17, 2010, 8:33 p.m. ET

Facebook in Privacy Breach

Top-Ranked Applications Transmit Personal IDs, a Journal Investigation Finds

Web platforms are **scary**!

They allow third-party developers to build apps that use our personal data.

Symantec. | Connect

COMMUNITY: Security | Blogs | Security Response

Facebook Applications Accidentally Leaking Access to Third Parties - Updated

The GitHub Blog

March 4, 2012 mojombo

Public Key Security Vulnerability

Trust concerns

- Don’t know the developers
  - Cannot determine trustworthiness of apps
- They may be malicious or security-unaware
- Building secure web apps is hard
  - Even well-meaning authors cannot be trusted

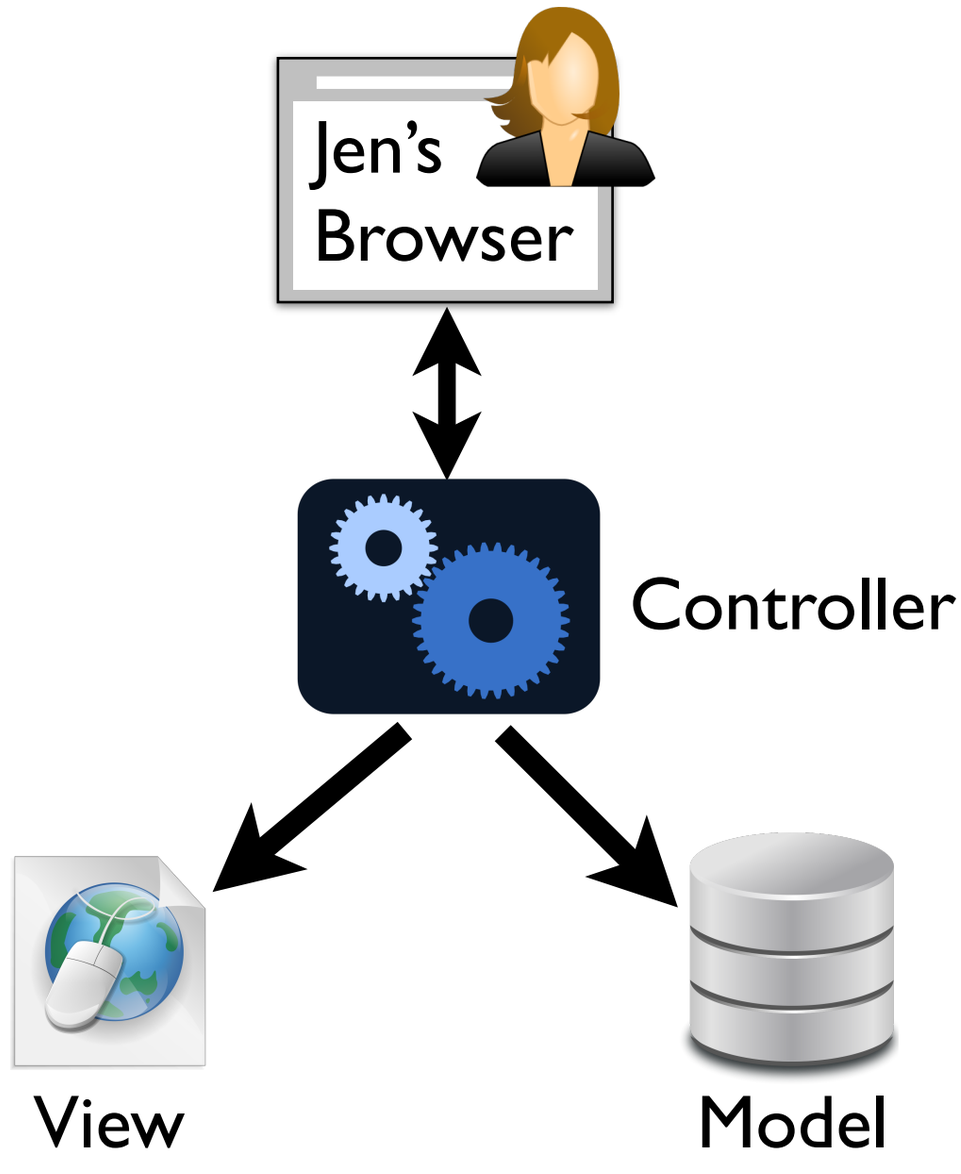
Typical App Design

Use popular MVC paradigm

Model: interface to data

View: renders pages

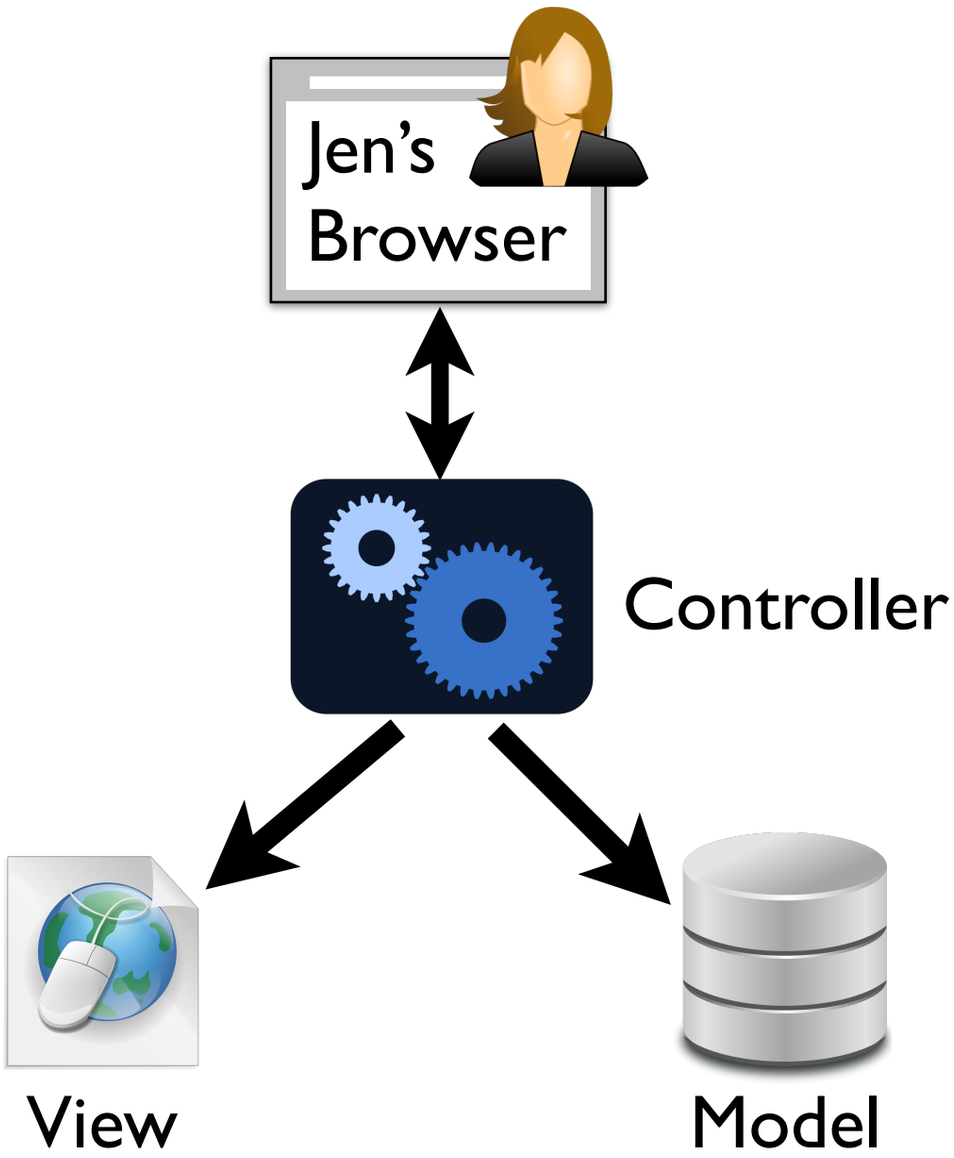
Controller: handles and responds to HTTP requests



Typical App Design

How is security policy specified and enforced?

- E.g., *only Jen's friends may see her email address*



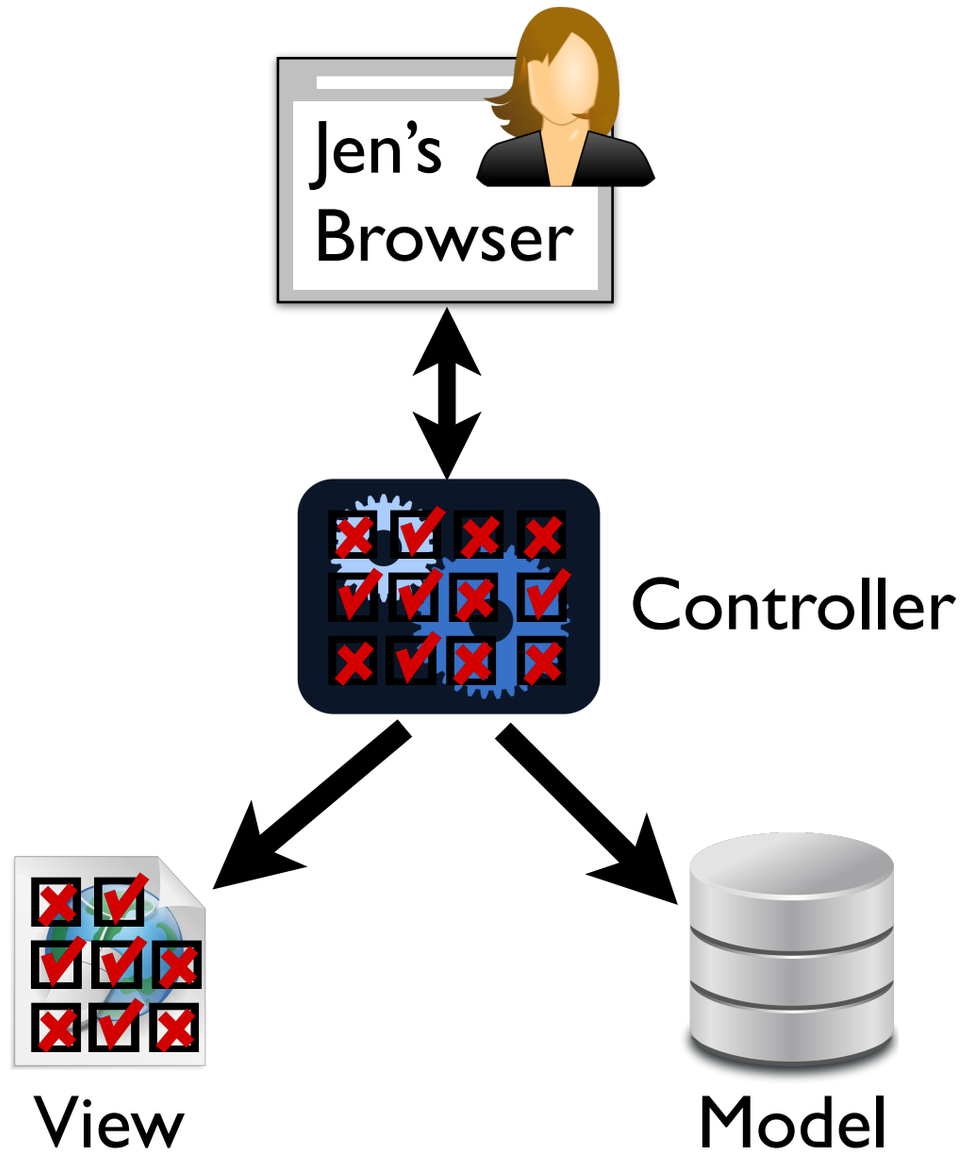
Typical App Design

How is security policy specified and enforced?

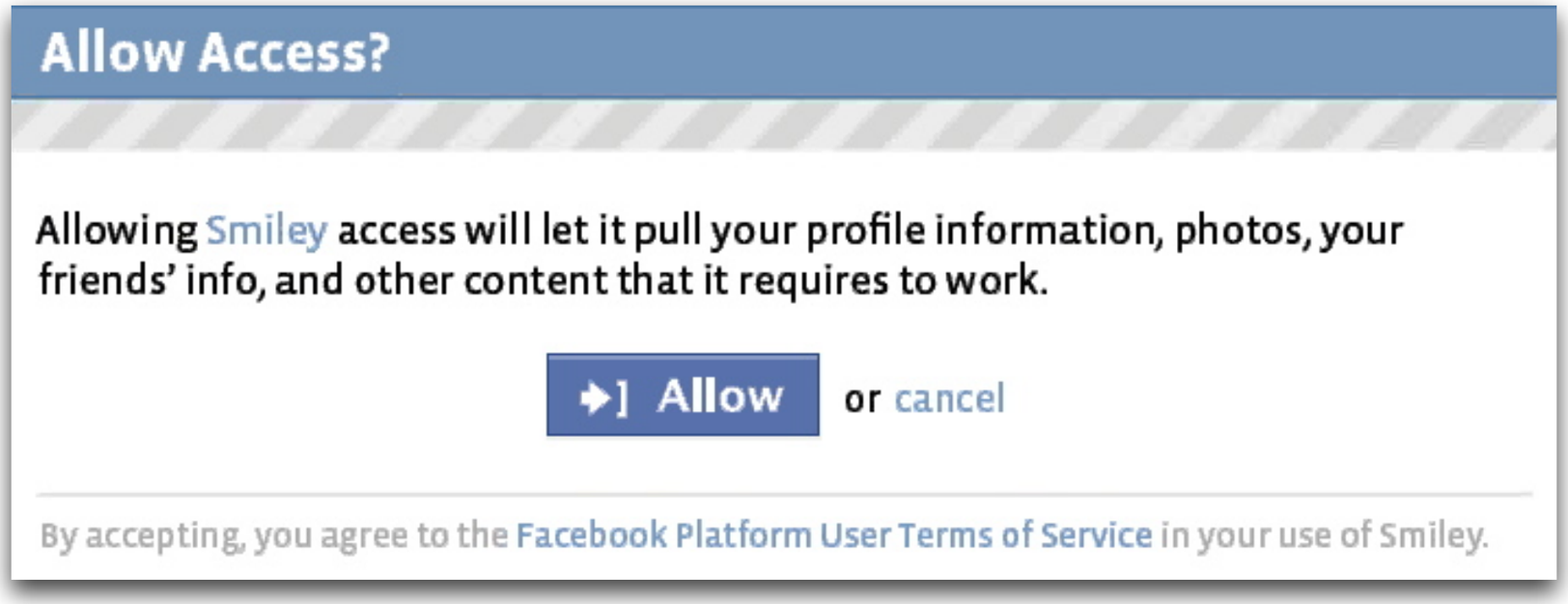
- E.g., *only Jen's friends may see her email address*

Intertwined throughout code

➠ Error prone and not scalable



Platform ~~solutions~~



Can decide to give an app access to data, but can't control how the app uses your data.

PCMAG.COM

Path Uploads Your Entire iPhone Contact List By Default

THE WALL STREET JOURNAL.

Home | World | Europe | U.K. | U.S. | Business | Markets | Market Data

WHAT THEY KNOW | October 17, 2010, 8:33 p.m. ET

Facebook in Privacy Breach

Top-Ranked Applications Transmit Personal IDs, a Journal Investigation Finds

Is there any hope for privacy on platforms?

Symantec. | Connect

COMMUNITY: Security | Blogs | Security Response

Facebook Applications Accidentally Leaking Access to Third Parties - Updated

The GitHub Blog

March 4, 2012 mojombo

Public Key Security Vulnerability

Change the hosting model

- Current model
  - App developers host their own apps
  - Platform enforces security: terms of service
- New model
  - Platform provider hosts apps
  - Platform enforces security: information flow control

Hails: A web platform framework

- Security policy is explicit and first-class
  - Specified as single concise module
- Users still trust core platform components
- Apps are untrusted
  - Language-level information flow control guarantees apps always obey policy

What makes Hails different?

Aeolus, HiStar, Nexus, Jif, Ur/Web, ...

- No guide for structuring applications
- Policies are hard to write
- Not appropriate for dynamic systems, e.g., web
- Modify entire application stack

Goals

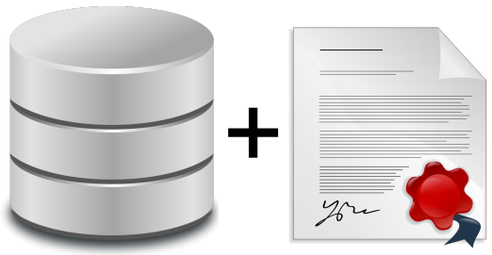
- Deployable today!
- Usable by web developers
- Suitable for building extensible web platforms
  - Enforcing policy across untrusted apps

Adding Policy to MVC

- New paradigm: Model-**Policy**-View-Controller
- Policy specified alongside data model
- No policy code in View or Controller

Two categories of code

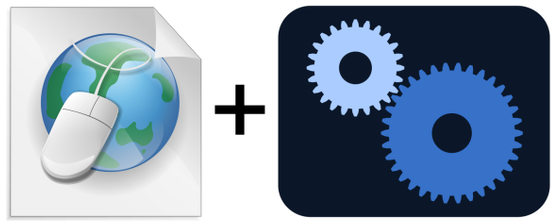
Models-Polices (MPs)



Specify data model and policy on data

- Users trust MPs they use to handle data

Views-Controllers (VCs)



Implement UI and other functionality

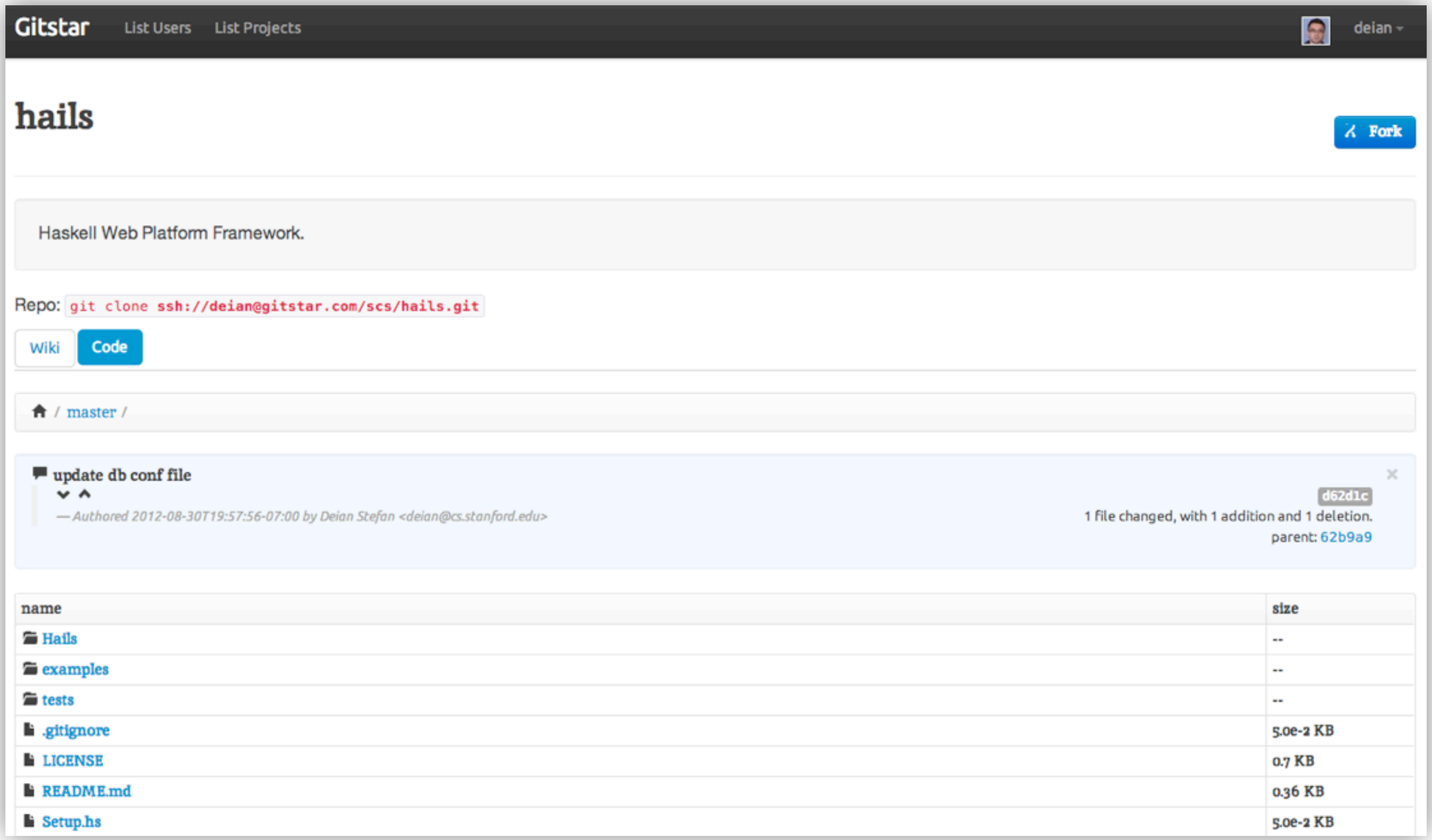
- Users need not trust VCs

Policy enforced globally

Information flow control

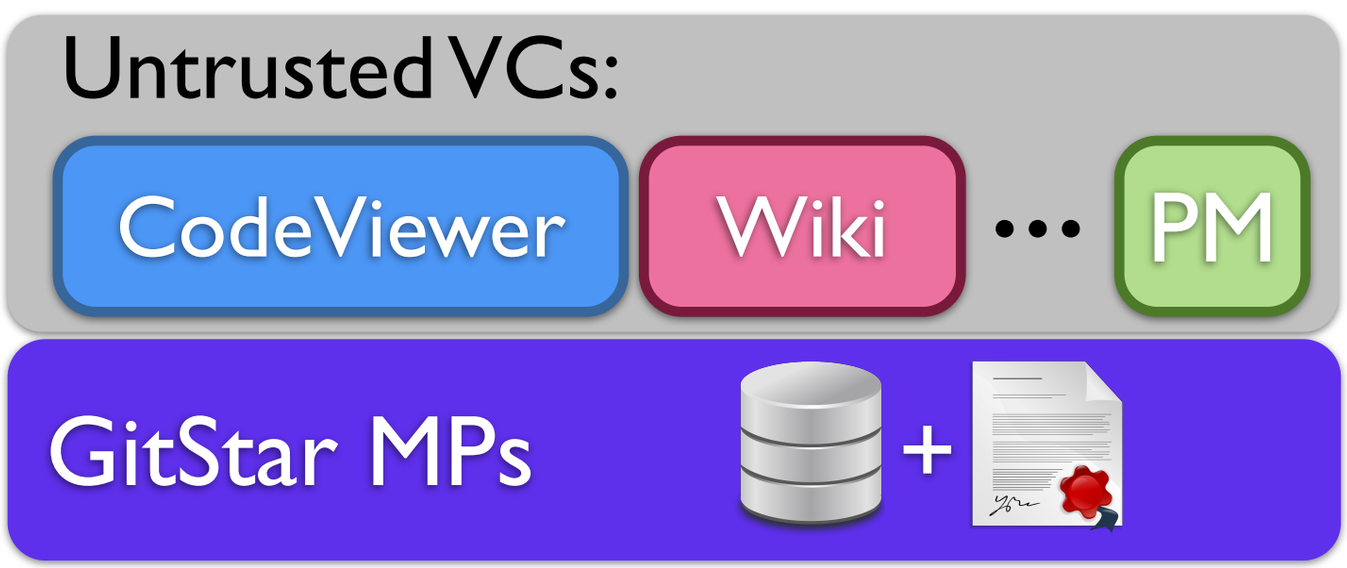
- Policy specifies where data can flow
  - **Wrong:** app can't read Jen's email address because it may leak it to Eve
  - **Right:** app can read Jen's email, but only reveal it to Jen, Alice or Bob
- Policy follows data through system
- Runtime enforces policy end-to-end
  - E.g., when making HTTP request

Case study: GitStar

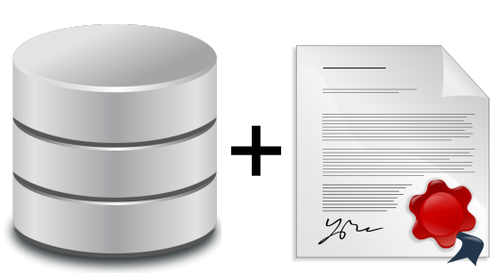


Case study: GitStar

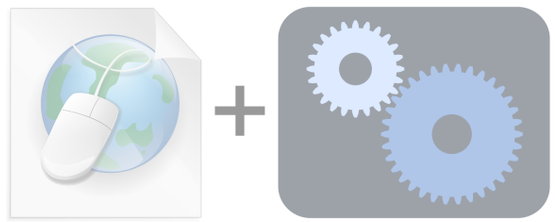
- GitStar provides
  - MPs that specify projects and users
  - VC for managing projects and users
- Third-party authors provide
  - Code viewer
  - Wiki
  - Follower app
  - etc.



Model-Policy



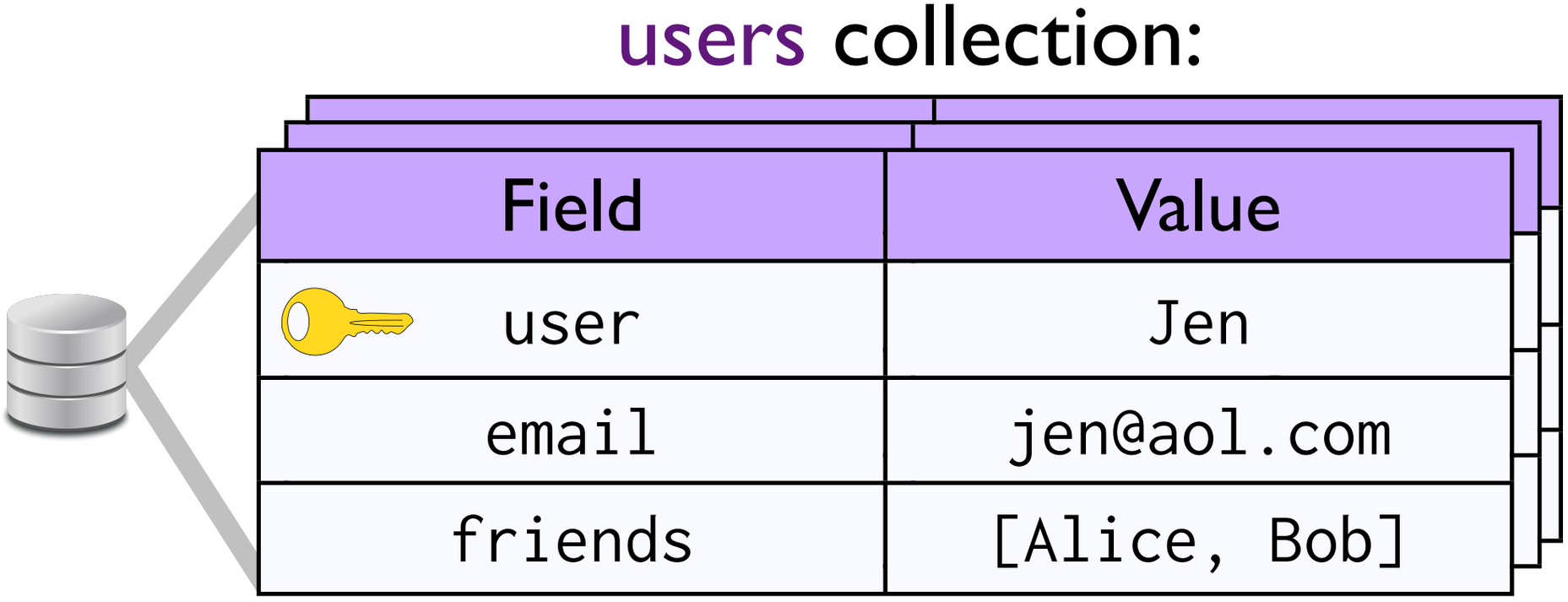
View-Controller



Model-Policy (MP)

Data model: document-oriented

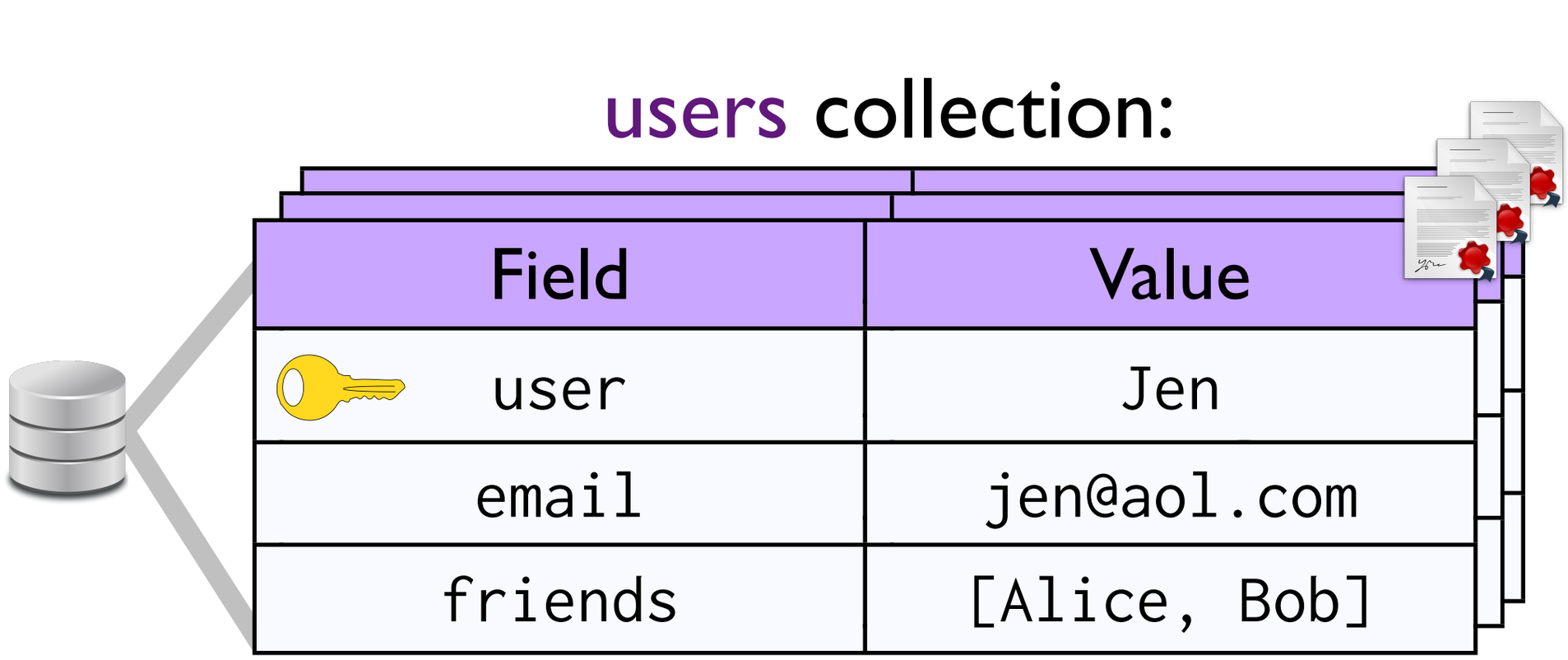
- **Collection:** set of documents
- **Document**: set of field-value pairs



Model-Policy (MP)

Data model: document-oriented

- **Collection:** set of documents
- **Document**: set of field-value pairs



Model-Policy (MP)

- Policy specifies restrictions on:
  - Collections, documents, fields
  - E.g., *only Jen may modify her profile*
  - E.g., *only Jen and her friends may read her email address*
- Policy composes
  - E.g., *to read document you must be able to read the collection*

Example: Enforcing policy

- MP:

Field	Value
user	Jen
email	jen@aol.com
friends	[Alice, Bob]

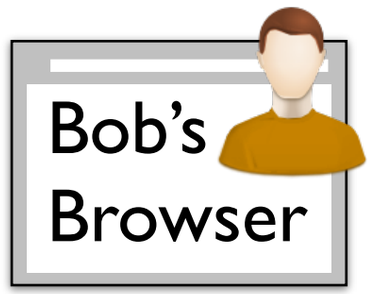
Policy: Only Jen, Alice and Bob can read

GitStar User MP +

- Eve's untrusted address book VC:

AddrBook

Example: Enforcing policy



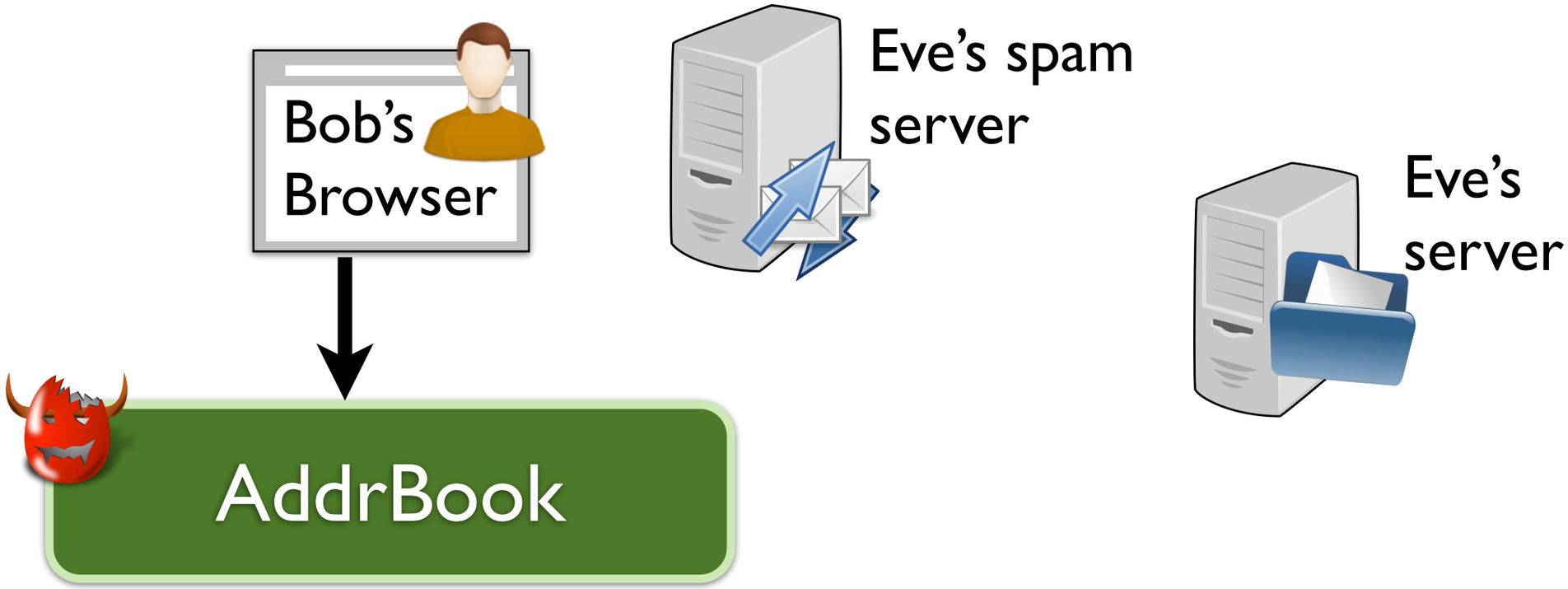
Eve's spam server

Eve's server

AddrBook

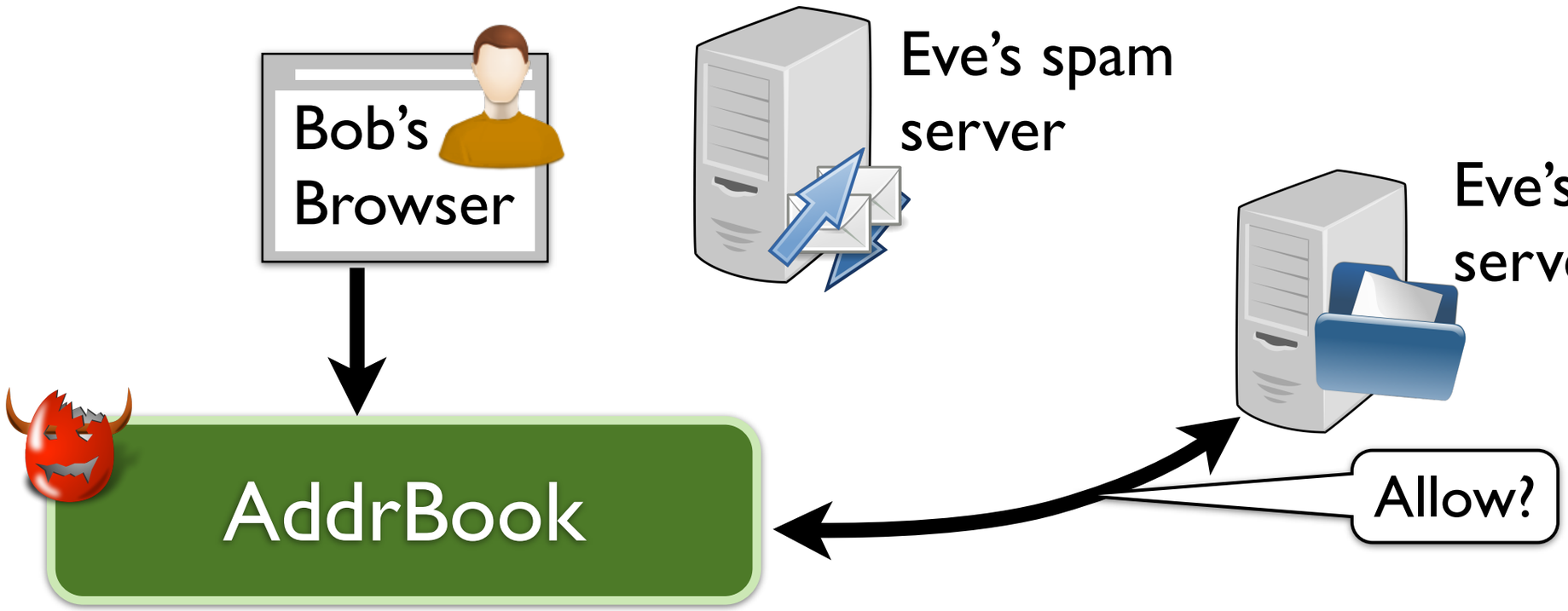
GitStar User MP  + 

Example: Enforcing policy



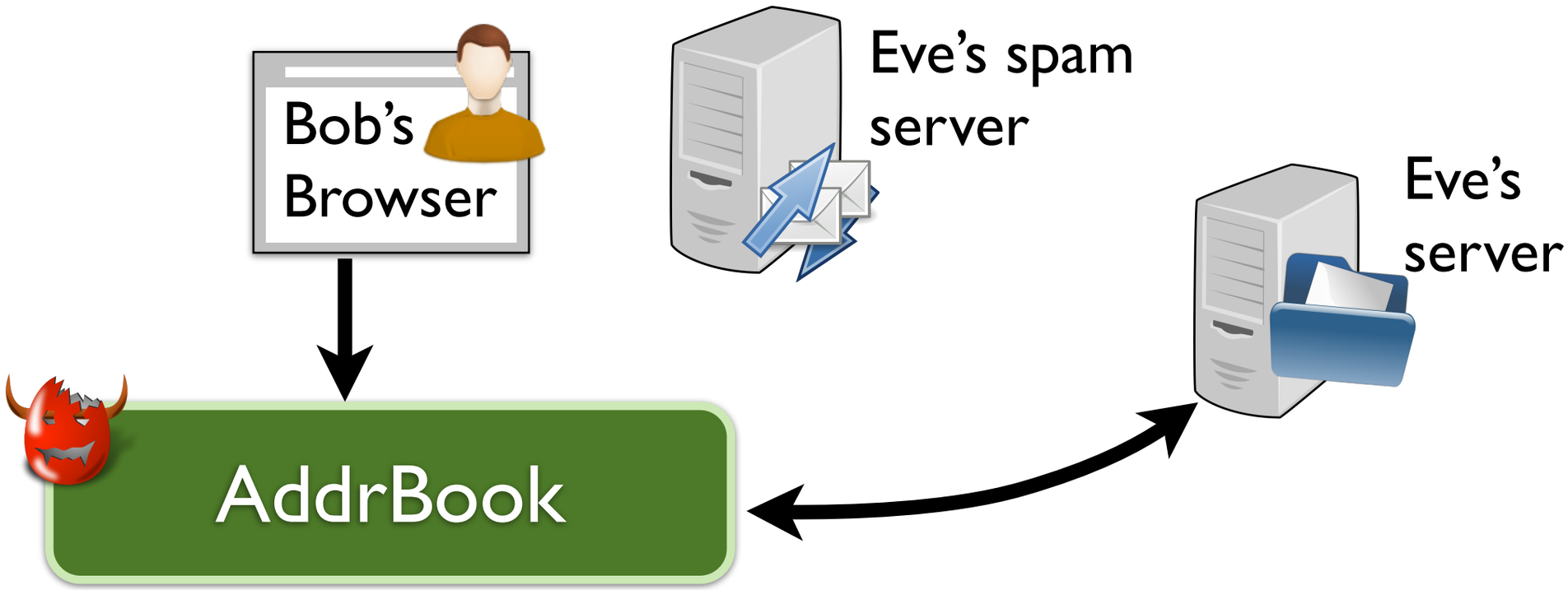
GitStar User MP + 

Example: Enforcing policy



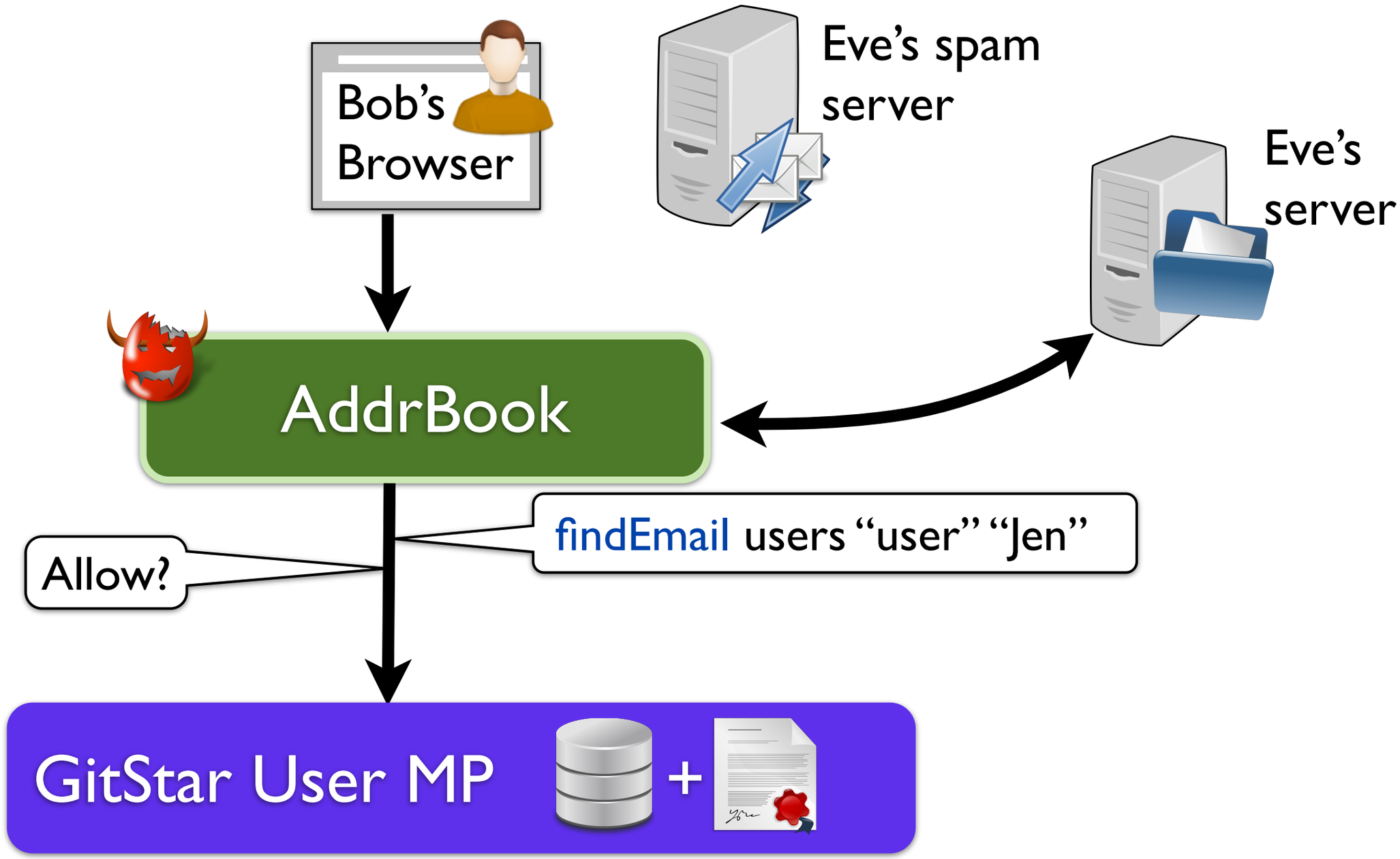
GitStar User MP + 

Example: Enforcing policy

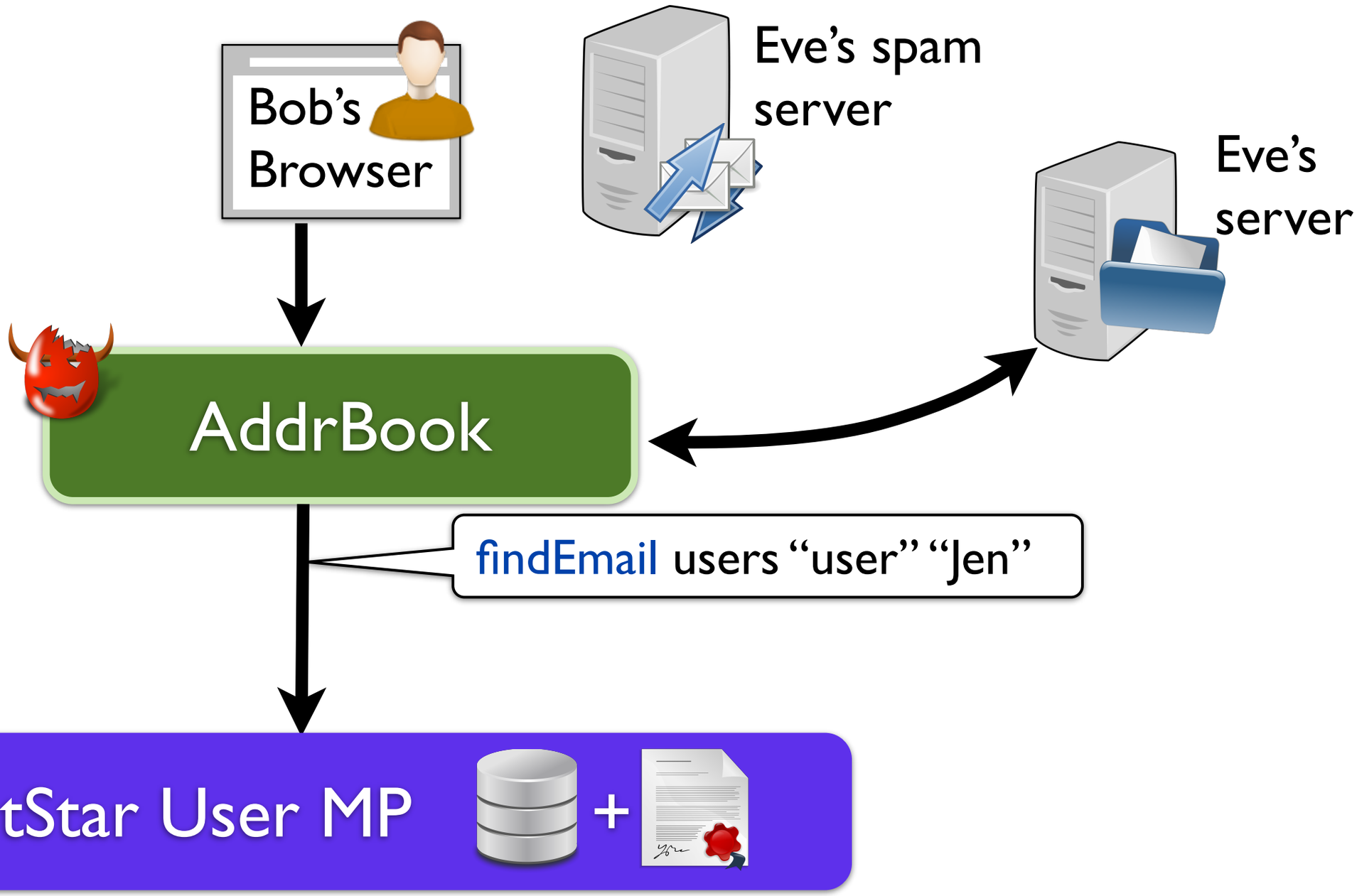


GitStar User MP + 

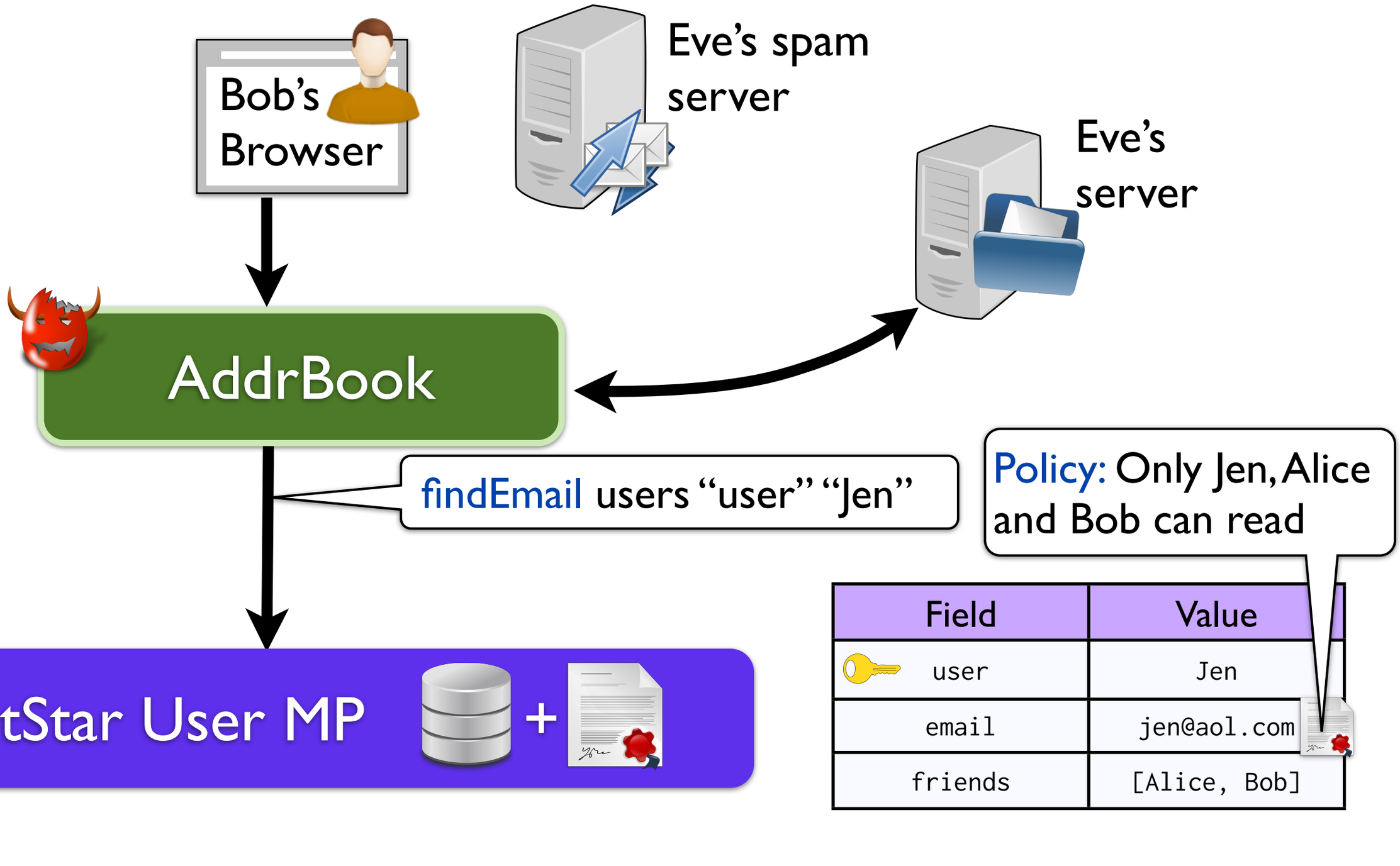
Example: Enforcing policy



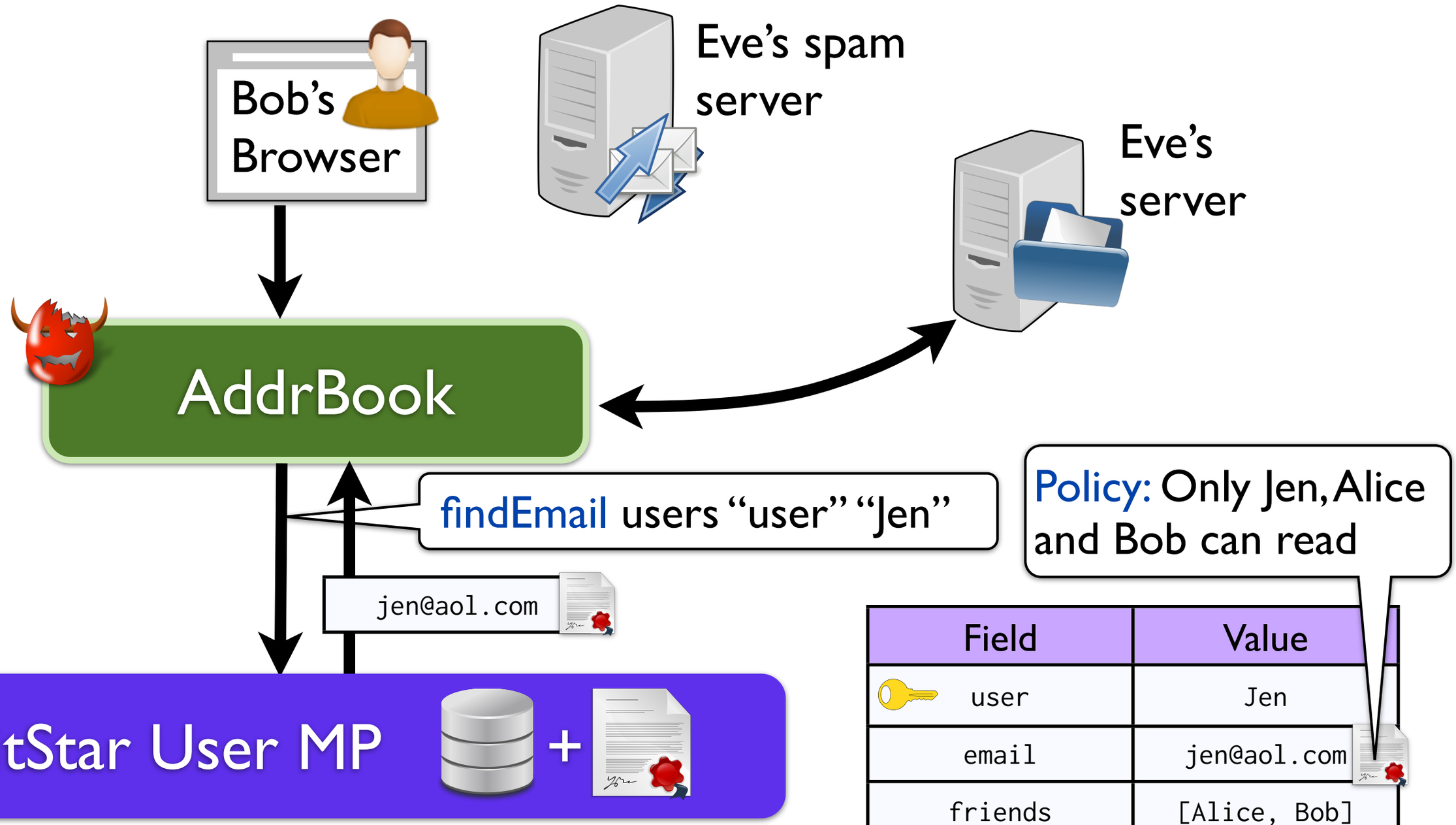
Example: Enforcing policy



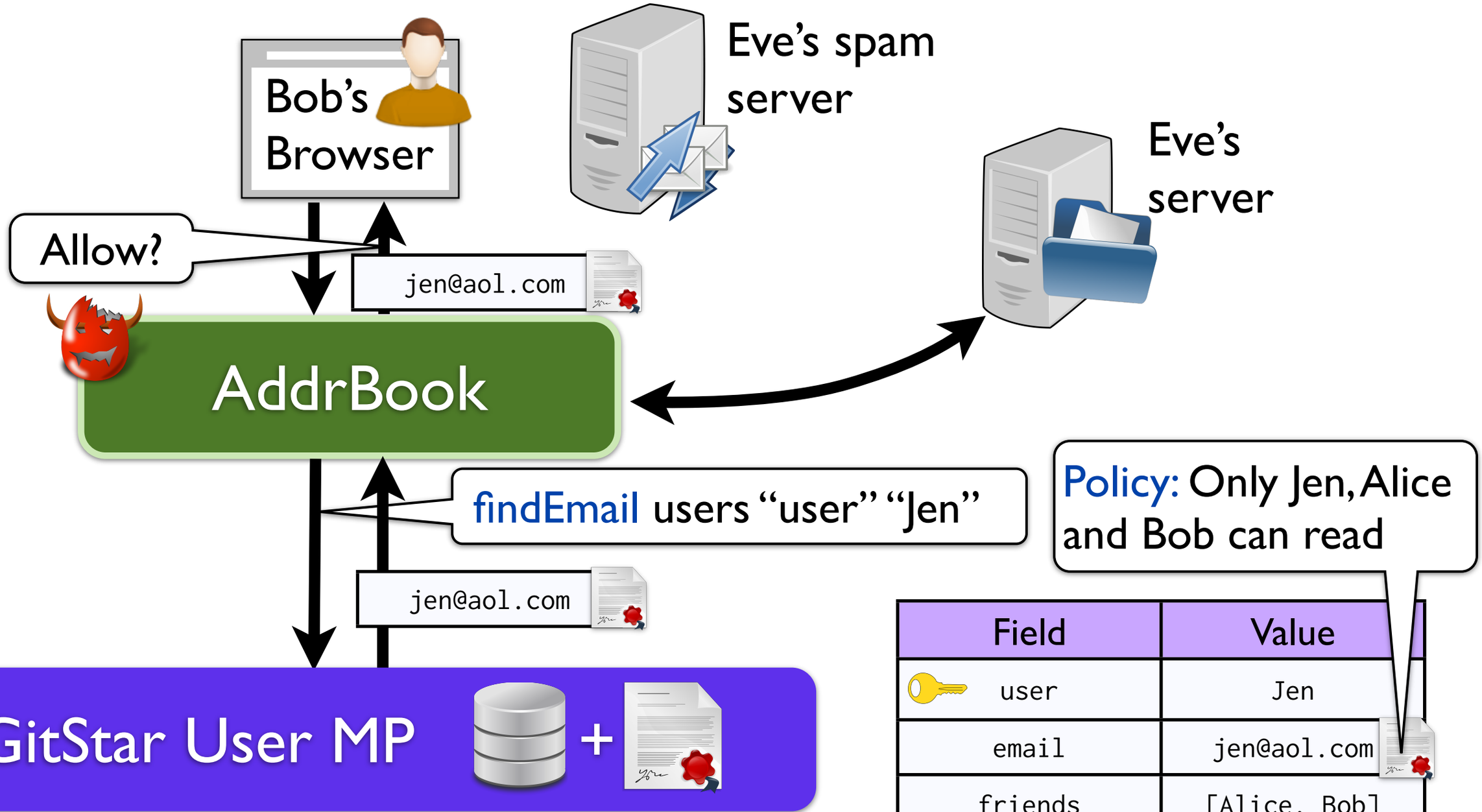
Example: Enforcing policy



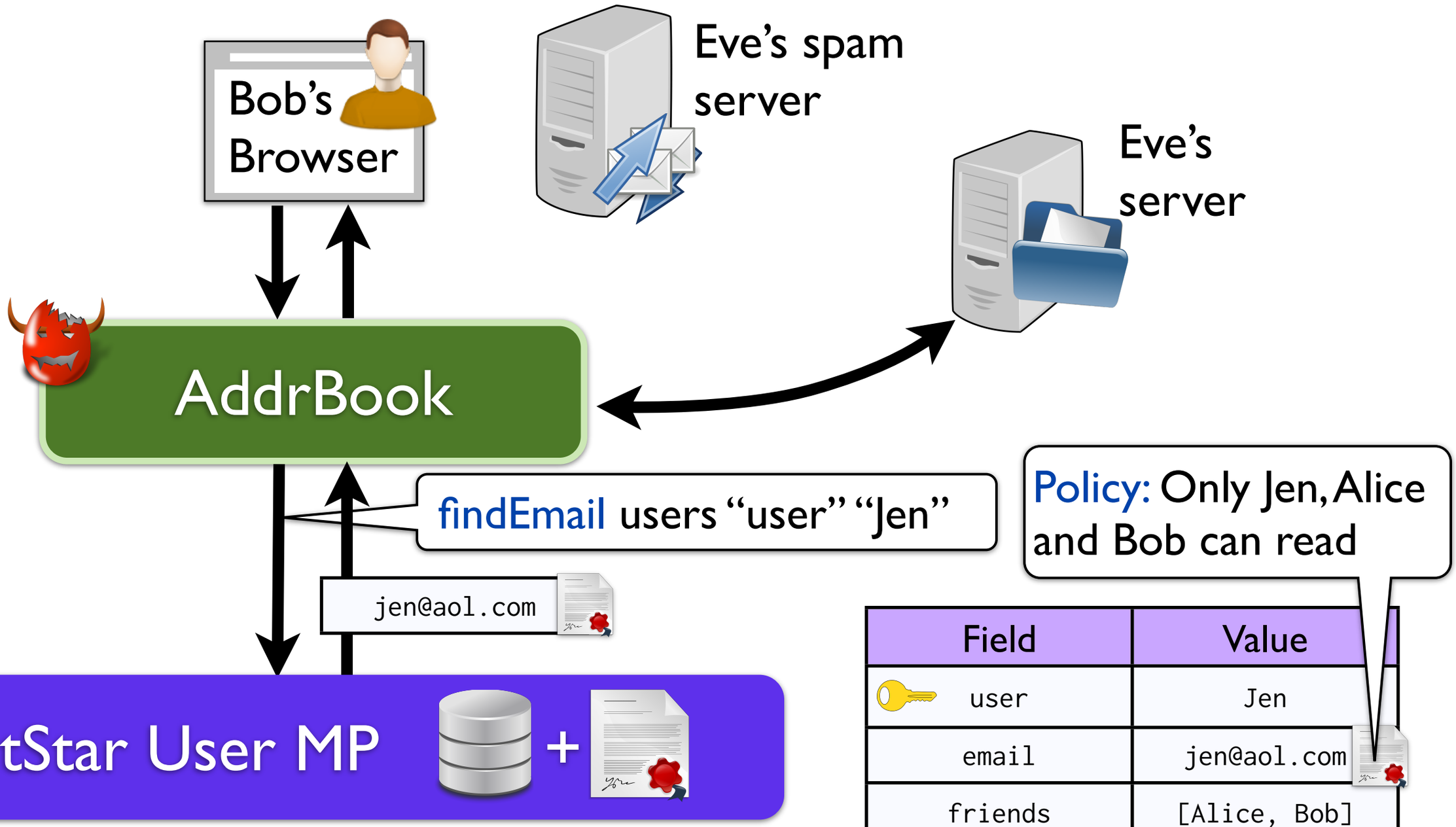
Example: Enforcing policy



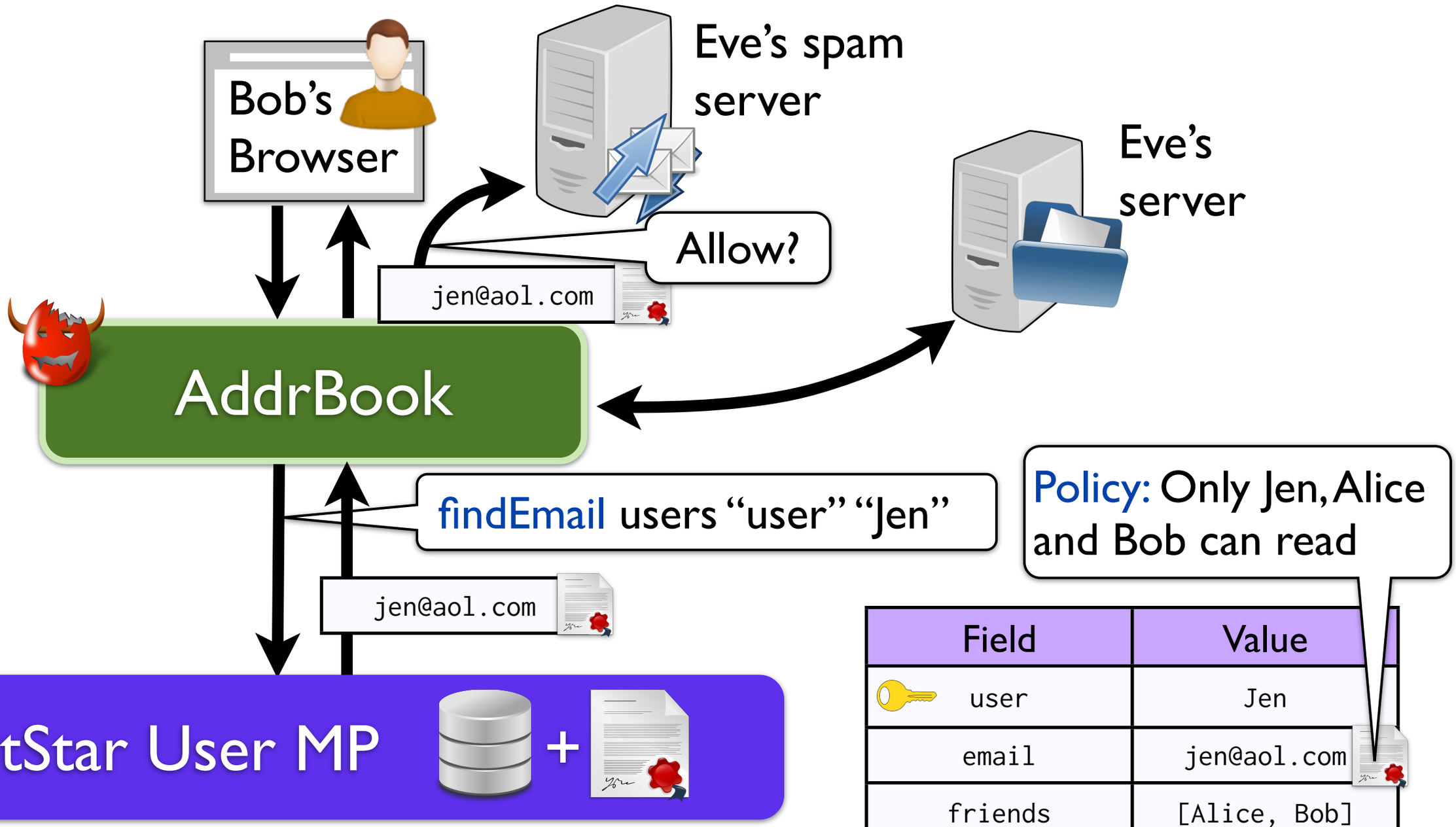
Example: Enforcing policy



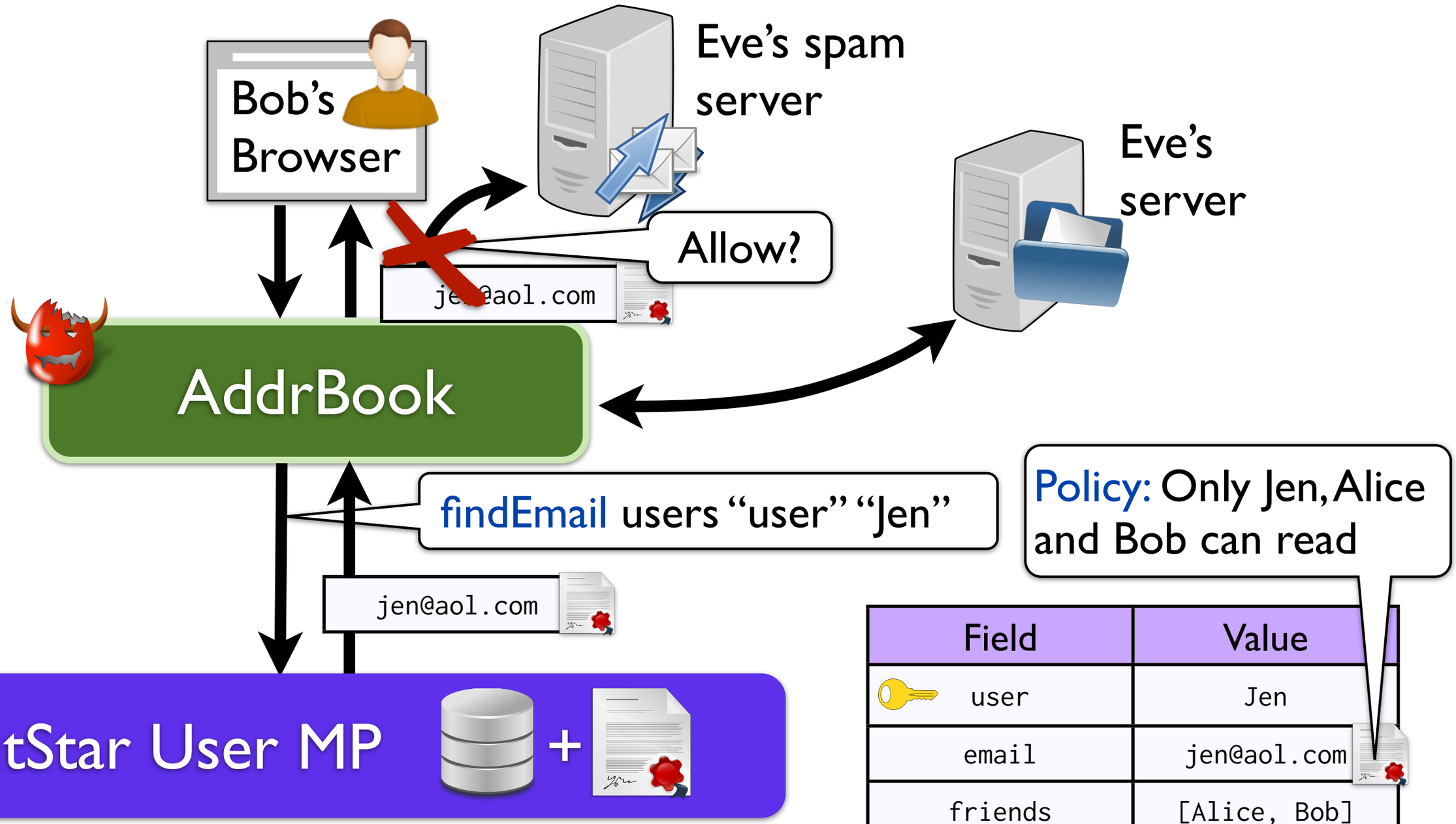
Example: Enforcing policy



Example: Enforcing policy



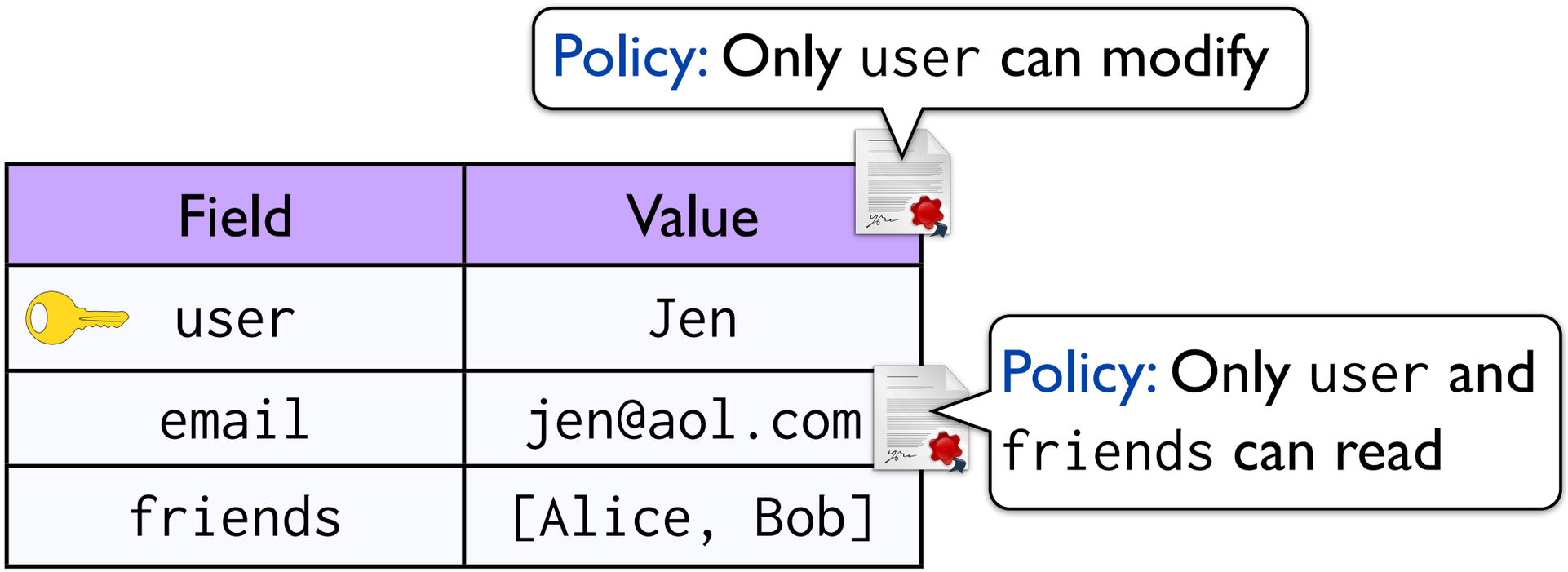
Example: Enforcing policy



Policy specified in terms of data

Web app data models already encode policy

- Ownership
- Relationships between users
- ...



Example: Policy specification

```
collection “users” $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field “user” key
    document $ λdoc -> do
      readers ==> anybody
      writers ==> (“user” `from` doc)
    field “email” $ labeled $ λdoc -> do
      readers ==> (“user” `from` doc)
                ∨ fromList (“friends” `from` doc)
      writers ==> anybody
```

Example: Policy specification

```
collection “users” $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field “user” key
    document $ λdoc -> do
      readers ==> anybody
      writers ==> (“user” `from` doc)
    field “email” $ labeled $ λdoc -> do
      readers ==> (“user” `from` doc)
                ∨ fromList (“friends” `from` doc)
      writers ==> anybody
```

Collection is public

Example: Policy specification

```
collection “users” $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field “user” key
    document $ λdoc -> do
      readers ==> anybody
      writers ==> (“user” `from` doc)
    field “email” $ labeled $ λdoc -> do
      readers ==> (“user” `from` doc)
                ∨ fromList (“friends” `from` doc)
      writers ==> anybody
```

Example: Policy specification

```
collection “users” $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field “user” key
    document $ λdoc -> do
      readers ==> anybody
      writers ==> (“user” `from` doc)
    field “email” $ labeled $ λdoc -> do
      readers ==> (“user” `from` doc)
              ∨ fromList (“friends” `from` doc)
      writers ==> anybody
```

Index documents by user names

Example: Policy specification

```
collection “users” $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field “user” key
    document $ λdoc -> do
      readers ==> anybody
      writers ==> (“user” `from` doc)
    field “email” $ labeled $ λdoc -> do
      readers ==> (“user” `from` doc)
                ∨ fromList (“friends” `from` doc)
      writers ==> anybody
```

Example: Policy specification

```
collection “users” $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field “user” key
    document $ λdoc -> do
      readers ==> anybody
      writers ==> (“user” `from` doc)
    field “email” $ labeled $ λdoc -> do
      readers ==> (“user” `from` doc)
              ∨ fromList (“friends” `from` doc)
      writers ==> anybody
```

Only Jen can modify document fields

Example: Policy specification

```
collection “users” $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field “user” key
    document $ λdoc -> do
      readers ==> anybody
      writers ==> (“user” `from` doc)
    field “email” $ labeled $ λdoc -> do
      readers ==> (“user” `from` doc)
                ∨ fromList (“friends” `from` doc)
      writers ==> anybody
```

Example: Policy specification

```
collection “users” $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field “user” key
document $ λdoc -> do
    readers ==> anybody
    writers ==> (“user” `from` doc)
field “email” $ labeled $ λdoc -> do
    readers ==> (“user” `from` doc)
            ∨ fromList (“friends” `from` doc)
    writers ==> anybody
```

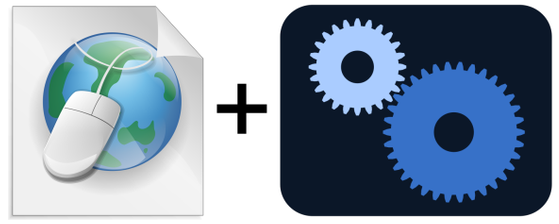
Only Jen, Alice and Bob can read Jen's email address

Example: Policy specification

```
collection “users” $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field “user” key
    document $ λdoc -> do
      readers ==> anybody
      writers ==> (“user” `from` doc)
    field “email” $ labeled $ λdoc -> do
      readers ==> (“user” `from` doc)
                ∨ fromList (“friends” `from` doc)
      writers ==> anybody
```

Model-Policy

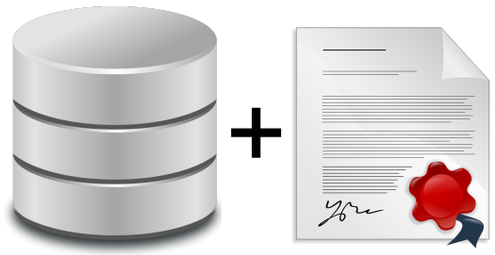
View-Controller



View-Controller (VC)

- A VC is a request handler
- Provide application functionality
  - E.g., source code browser, blog editor, …
- Invoke MPs to store/fetch user data
- Bugs in VCs are never vulnerabilities
  - Runtime enforces security policy

Model-Policy



View-Controller

Implications of MPVC

- Users: choose VCs based on functionality
- Devs: build apps on top of existing user-data
  - Models and policies are reusable

www.gitstar.com/scs/hails

Gitstar List Users List Projects deian

hails

Haskell Web Platform Framework.

Repo: `git clone ssh://deian@gitstar.com/scs/hails.git`

Wiki Code

Code viewer VC

/ master / examples / SimpleParams.hs

```
1.  {-# LANGUAGE OverloadedStrings #-}
2.  module SimpleParams (server) where
3.
4.  import qualified Data.ByteString.Lazy.Char8 as L8
5.
6.  import           LIO
7.  import           LIO.DCLabel
8.  import           Hails.HttpServer
9.  import           Hails.Data.Hson
10.
11. server :: Application
12. server _ lreq = do
13.   req <- unlabel lreq
14.   let ldoc = labeledRequestToLabeledDocument lreq
15.   doc <- unlabel ldoc
16.   return $ case pathInfo req of
17.     ("login":_) -> Response temporaryRedirect307
18.                            [("x-hails-login",""),(hLocation,"/")] ""
19.     _ -> Response ok200 [] $ topHtml (labelOf lreq, req) (labelOf ldoc, doc)
```

www.gitstar.com/scs/hails

Gitstar List Users List Projects

hails

Haskell Web Platform Framework.

Repo: `git clone ssh://deian@gitstar.com/scs/hails.git`

Wiki Code

Wiki VC

Home Pages

Hails: Protecting Data Privacy in Untrusted W

Hails is a platform and web framework that leverages Information Flow Control (IFC) to support u applications interacting and processing private data.

Resources

- Brief motivation and architecture overview
- Tutorial (slightly outdated)

Installation

You can compile and install Hails as usual with `cabal-dev`:

```
$ cabal-dev instal-deps
$ cabal-dev install
```

Launching an app

If you define your main application (named `server`) in `YourAppModule.hs`, you can launch it with

GitStar Project MP + 

Implementation

- Hails is a Haskell library
  - Quick turnaround on API design
  - Developers can use existing tools and libraries
- Hails runtime system
  - Provides HTTP server that invokes VC
  - Enforces information flow at the language-level

Evaluation: Usability

✓ MPVC simplifies reasoning about security when building a platform

✓ Hails renders common security bugs futile

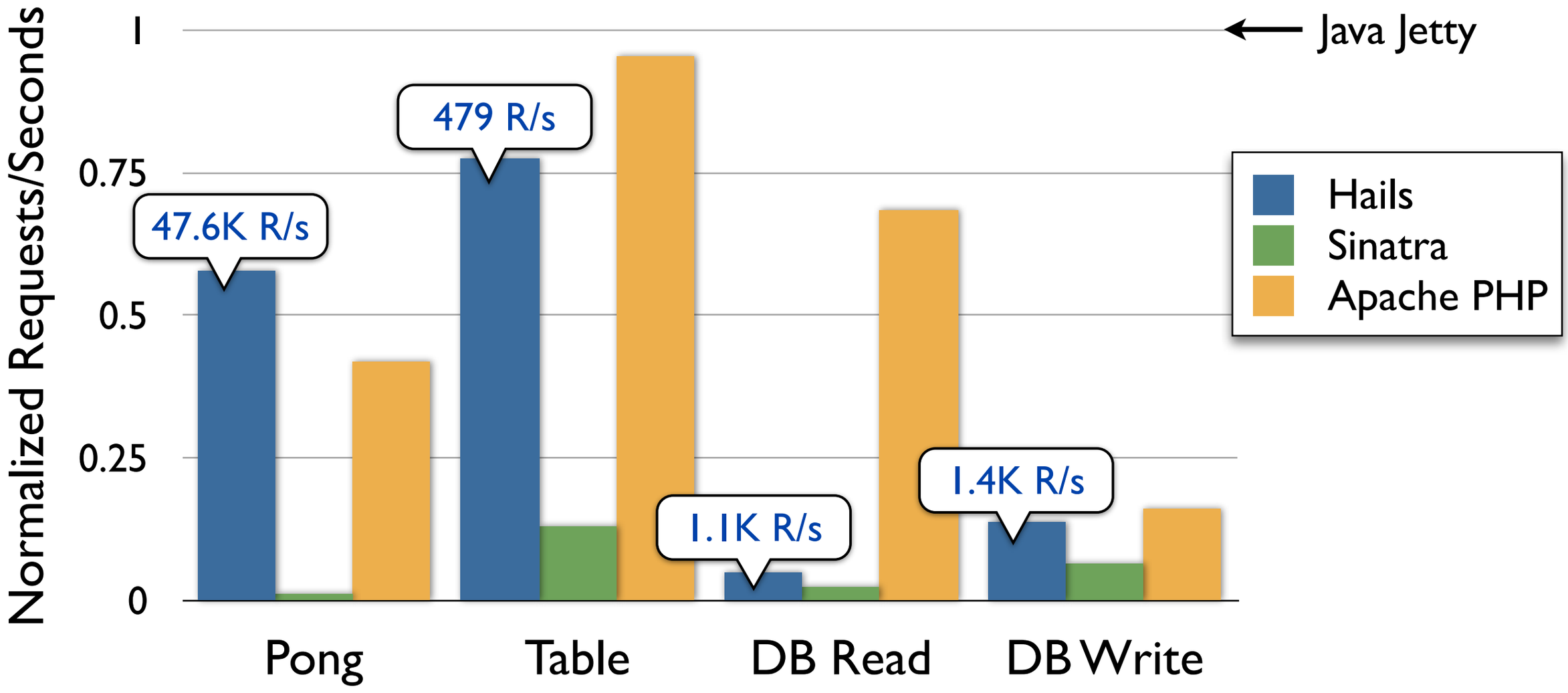
E.g., mass assignment vulnerability

✗ Need scaffolding tools

✗ ~~Writing raw policy is hard~~

✓ Writing policy with DSL is simpler

Performance evaluation



Conclusions

- Current platforms: functionality vs. privacy
- Hails platforms guarantee security across apps
  - Host apps on platform
  - Make policy explicit
  - Enforce policy with information flow control

http://gitstar.com http://hails.io

```
$ cabal install hails
```