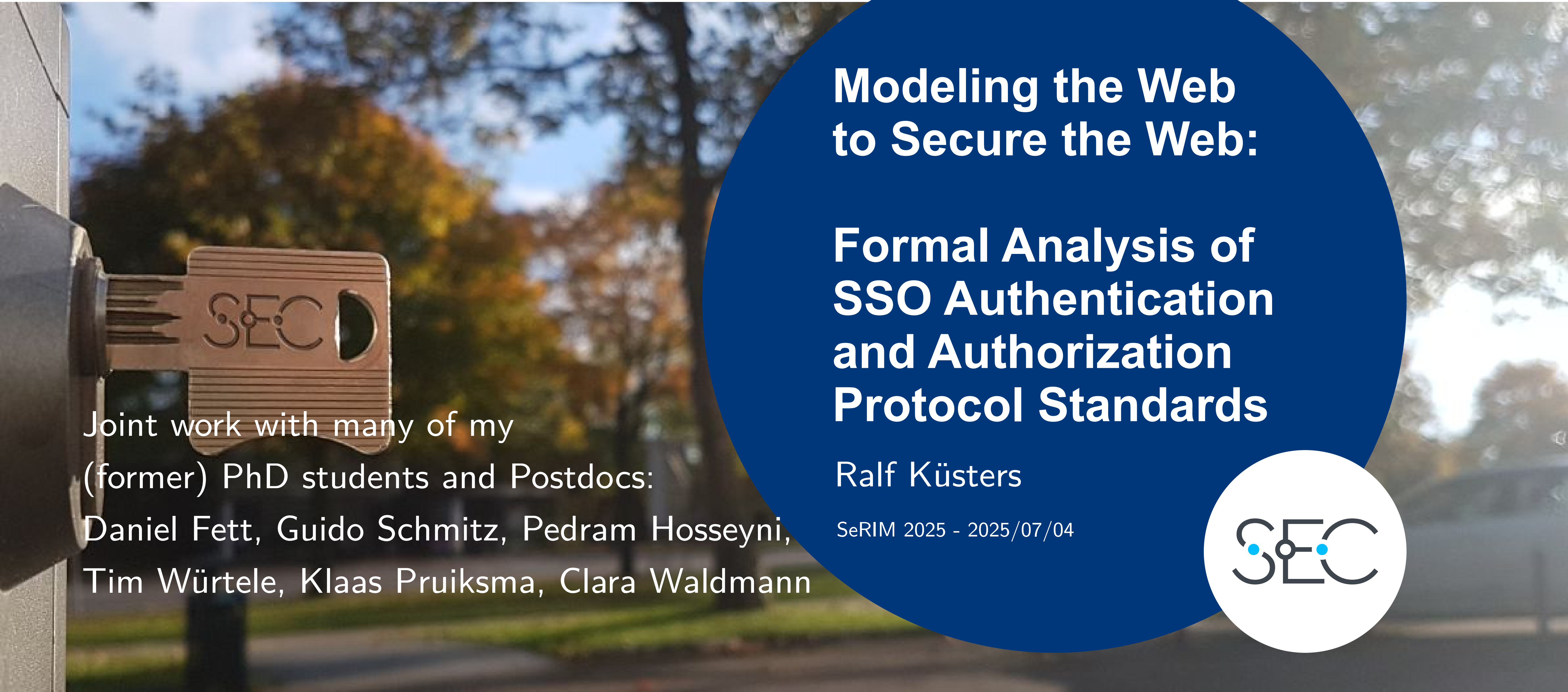
University of Stuttgart
Institute of
Information Security



Modeling the Web to Secure the Web: Formal Analysis of SSO Authentication and Authorization Protocol Standards

Ralf Küsters

SeRIM 2025 - 2025/07/04

Joint work with many of my (former) PhD students and Postdocs:
Daniel Fett, Guido Schmitz, Pedram Hosseyni, Tim Würtele, Klaas Pruiksma, Clara Waldmann

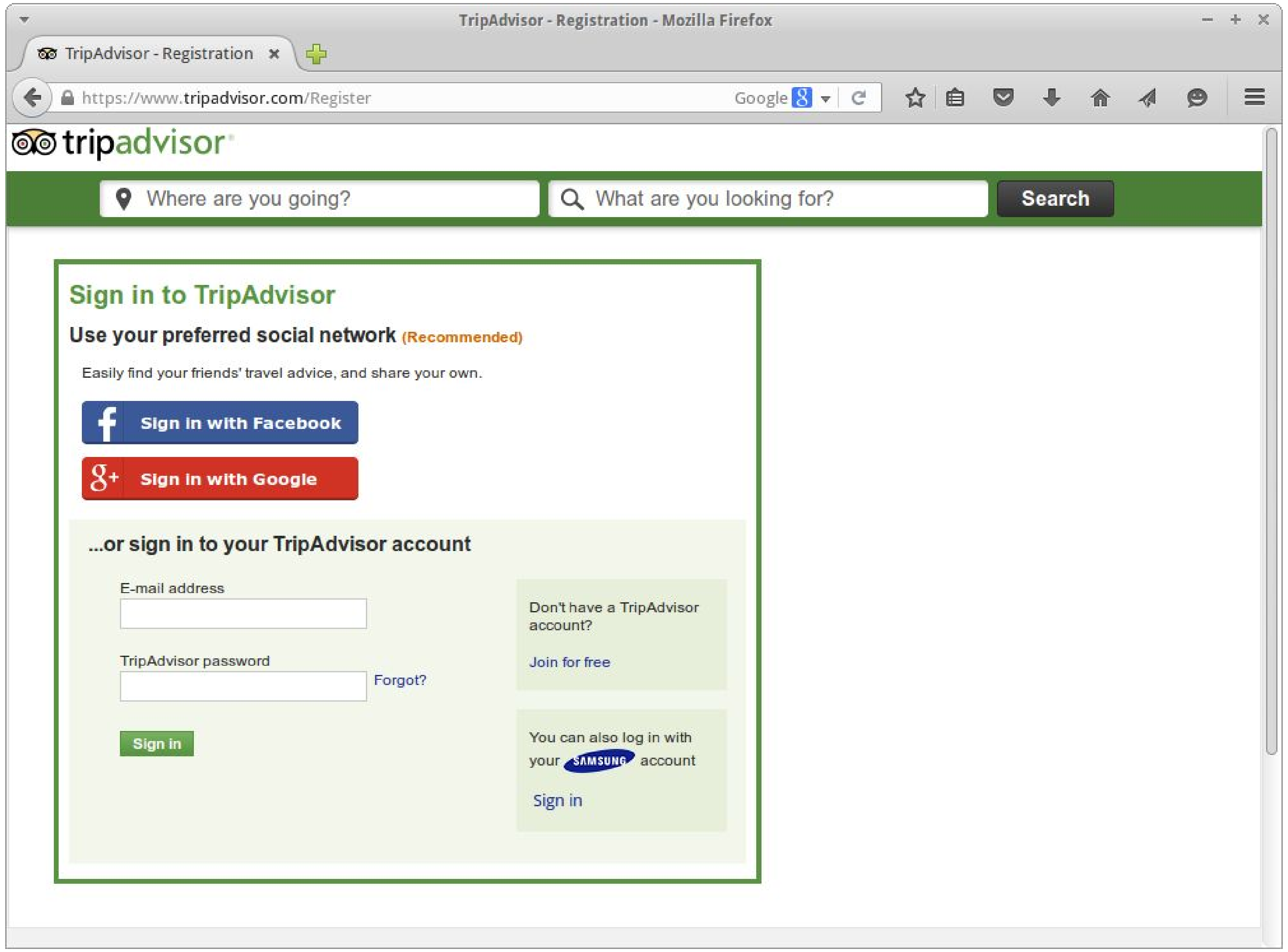
Single Sign-On (SSO): Social Web SSO – The Beginning

Single Sign-On (SSO): Social Web SSO – The Beginning

OAuth 2.0

OpenID Connect

Single Sign-On (SSO): Social Web SSO – The Beginning



TripAdvisor - Registration - Mozilla Firefox

TripAdvisor - Registration

https://www.tripadvisor.com/Register

tripadvisor

Where are you going? What are you looking for? Search

Sign in to TripAdvisor

Use your preferred social network (Recommended)

Easily find your friends' travel advice, and share your own.

Sign in with Facebook

Sign in with Google

...or sign in to your TripAdvisor account

E-mail address

TripAdvisor password Forgot?

Sign in

Don't have a TripAdvisor account?

Join for free

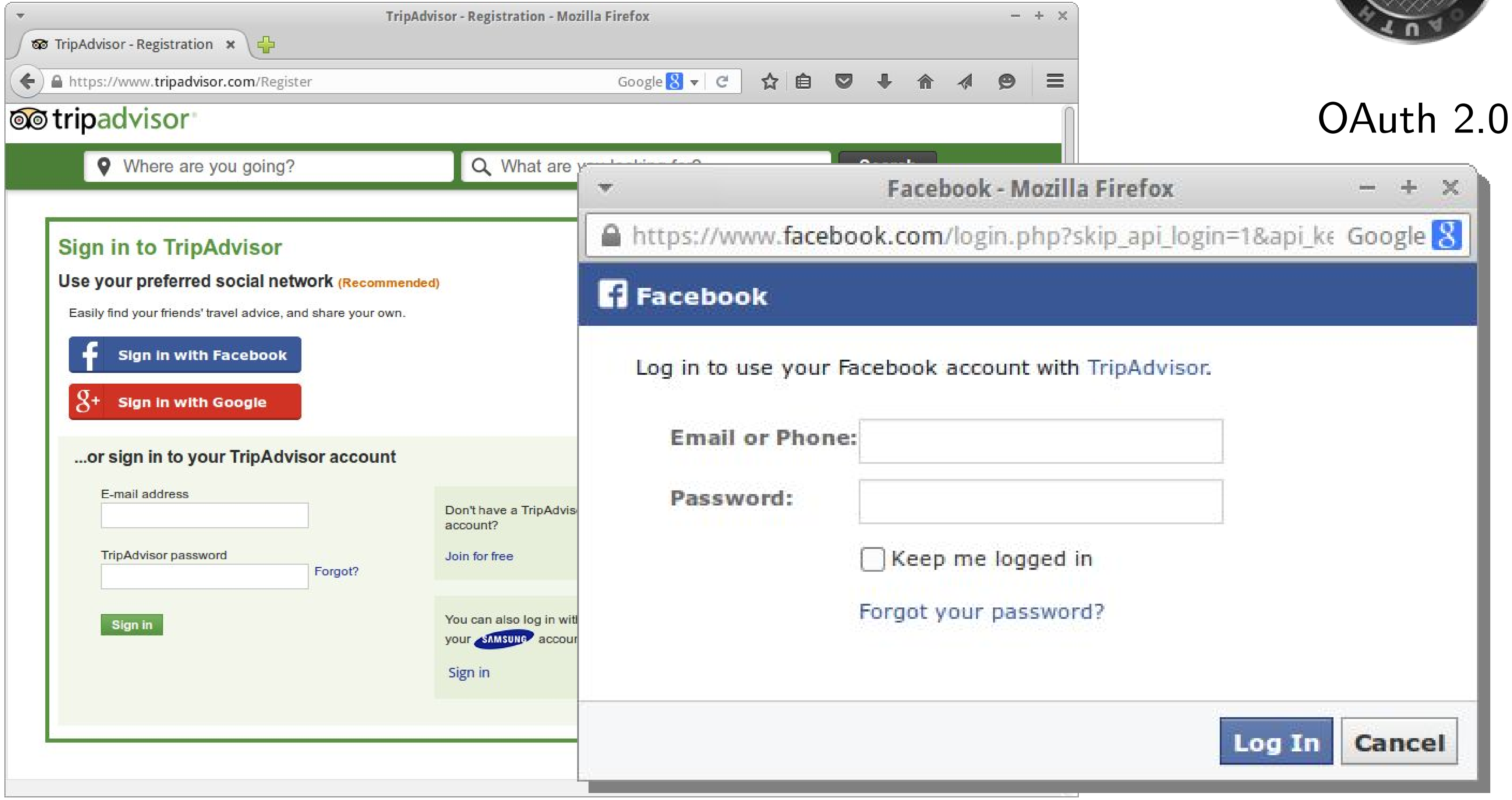
You can also log in with your SAMSUNG account

Sign in

OAuth 2.0

OpenID Connect

Single Sign-On (SSO): Social Web SSO – The Beginning


TripAdvisor - Registration - Mozilla Firefox

TripAdvisor - Registration

https://www.tripadvisor.com/Register

tripadvisor

Where are you going?

What are you looking for?

Sign in to TripAdvisor

Use your preferred social network (Recommended)

Easily find your friends' travel advice, and share your own.

Sign in with Facebook

Sign in with Google

...or sign in to your TripAdvisor account

E-mail address

TripAdvisor password

Forgot?

Sign in

Don't have a TripAdvis
account?

Join for free

You can also log in wit
your SAMSUNG accou

Sign in

Facebook - Mozilla Firefox

https://www.facebook.com/login.php?skip_api_login=1&api_ke

Facebook

Log in to use your Facebook account with TripAdvisor.

Email or Phone:

Password:

Keep me logged in

Forgot your password?

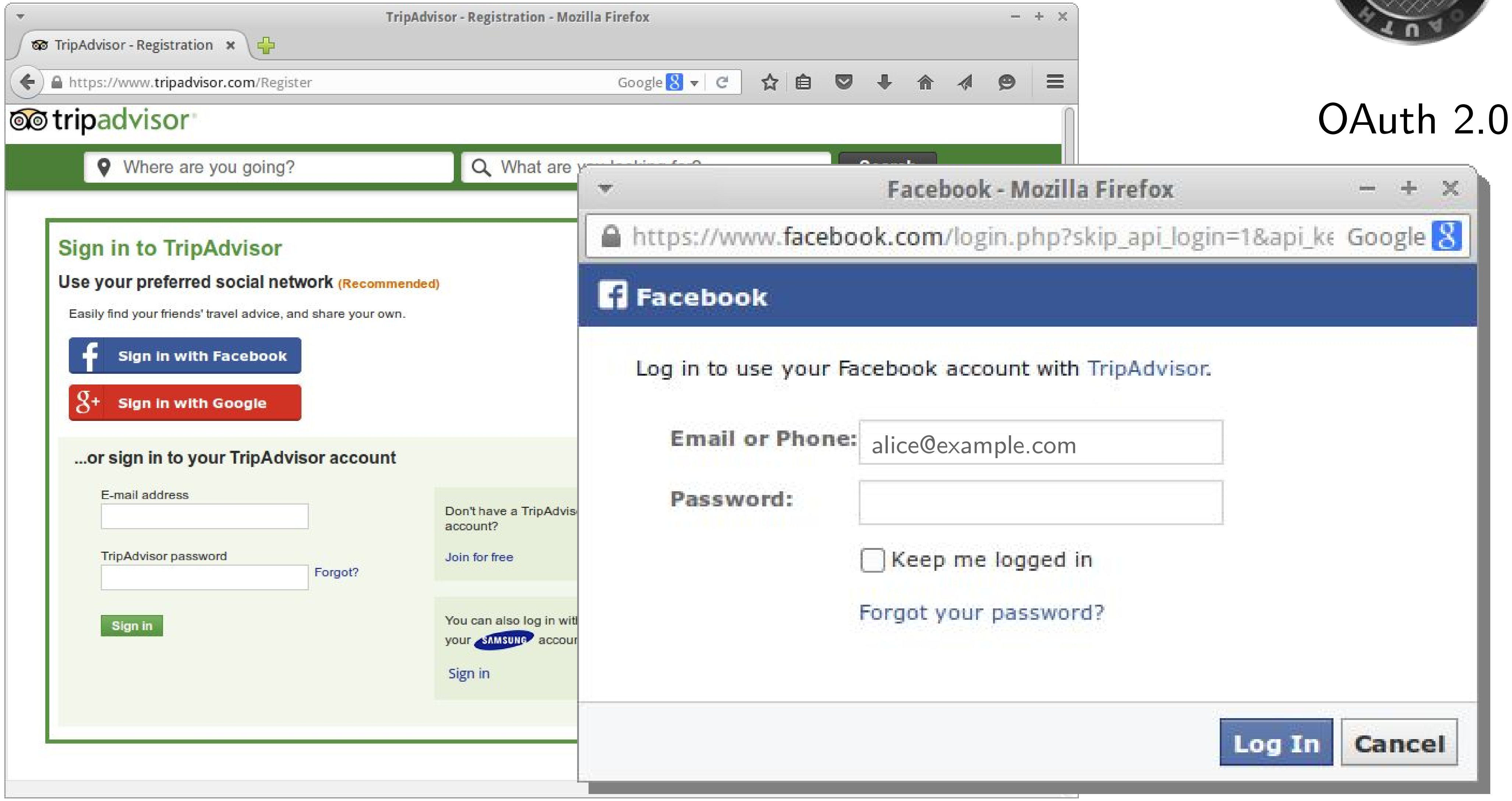
Log In

Cancel

OAuth 2.0

OpenID Connect

Single Sign-On (SSO): Social Web SSO – The Beginning



TripAdvisor - Registration - Mozilla Firefox

https://www.tripadvisor.com/Register

Sign in to TripAdvisor

Use your preferred social network (Recommended)

Easily find your friends' travel advice, and share your own.

Sign in with Facebook

Sign in with Google

...or sign in to your TripAdvisor account

E-mail address

TripAdvisor password

Forgot?

Sign in

Don't have a TripAdvis account?

Join for free

You can also log in wit your SAMSUNG accour

Sign in

Facebook - Mozilla Firefox

https://www.facebook.com/login.php?skip_api_login=1&api_ke

Facebook

Log in to use your Facebook account with TripAdvisor.

Email or Phone: alice@example.com

Password:

Keep me logged in

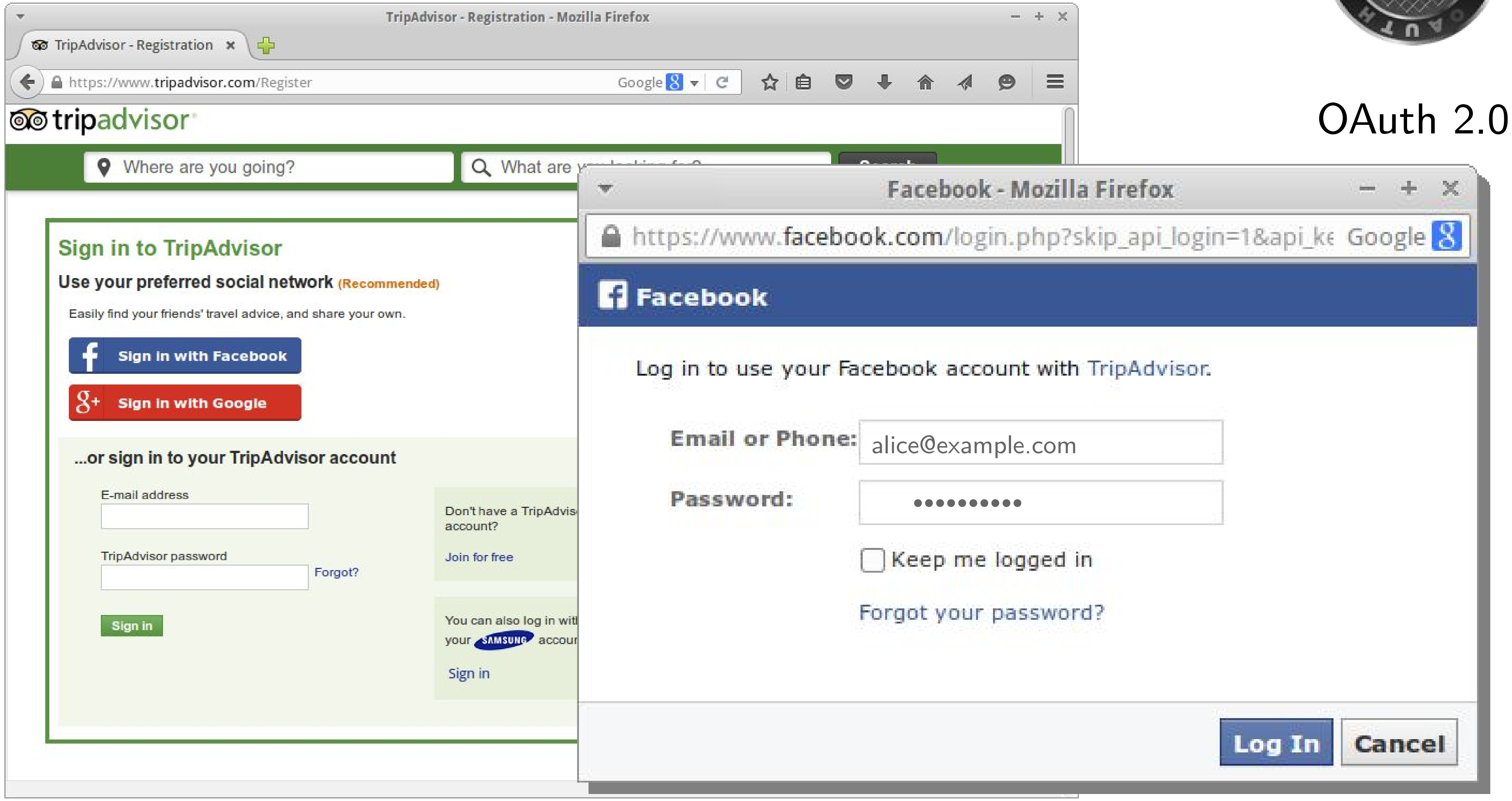
Forgot your password?

Log In Cancel

OAuth 2.0

OpenID Connect

Single Sign-On (SSO): Social Web SSO – The Beginning


TripAdvisor - Registration - Mozilla Firefox

https://www.tripadvisor.com/Register

Sign in to TripAdvisor

Use your preferred social network (Recommended)

Easily find your friends' travel advice, and share your own.

Sign in with Facebook

Sign in with Google

...or sign in to your TripAdvisor account

E-mail address

TripAdvisor password

Forgot?

Sign in

Don't have a TripAdvis account?

Join for free

You can also log in wit your SAMSUNG accour

Sign in

Facebook - Mozilla Firefox

https://www.facebook.com/login.php?skip_api_login=1&api_ke

Facebook

Log in to use your Facebook account with TripAdvisor.

Email or Phone:	alice@example.com
Password:	●●●●●●●●●●●

Keep me logged in

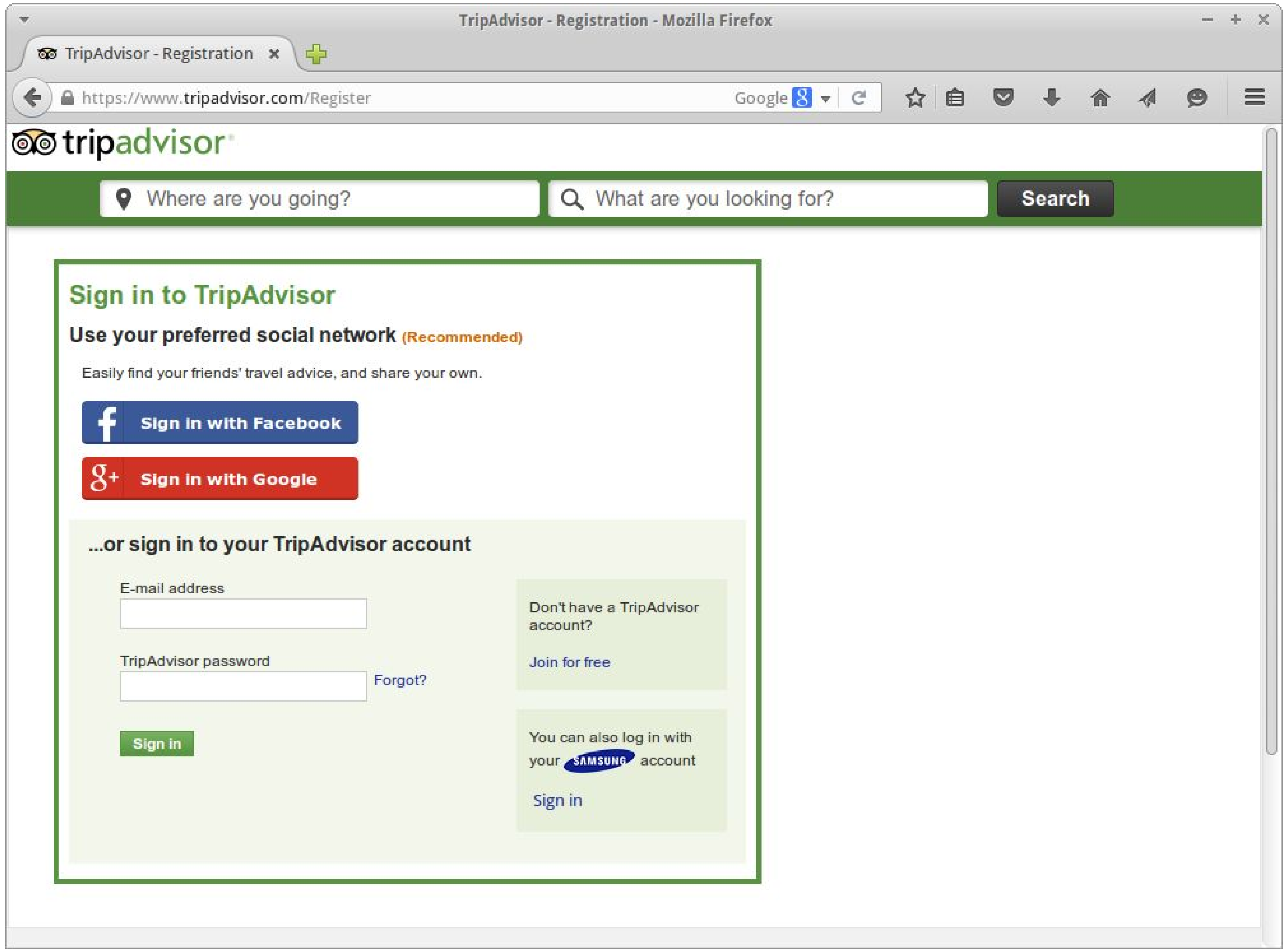
Forgot your password?

Log In Cancel

OAuth 2.0

OpenID Connect

Single Sign-On (SSO): Social Web SSO – The Beginning


TripAdvisor - Registration - Mozilla Firefox

TripAdvisor - Registration

https://www.tripadvisor.com/Register

tripadvisor

Where are you going? | What are you looking for? | Search

Sign in to TripAdvisor

Use your preferred social network (Recommended)

Easily find your friends' travel advice, and share your own.

Sign in with Facebook

Sign in with Google

...or sign in to your TripAdvisor account

E-mail address

TripAdvisor password

Forgot?

Sign in

Don't have a TripAdvisor account?

Join for free

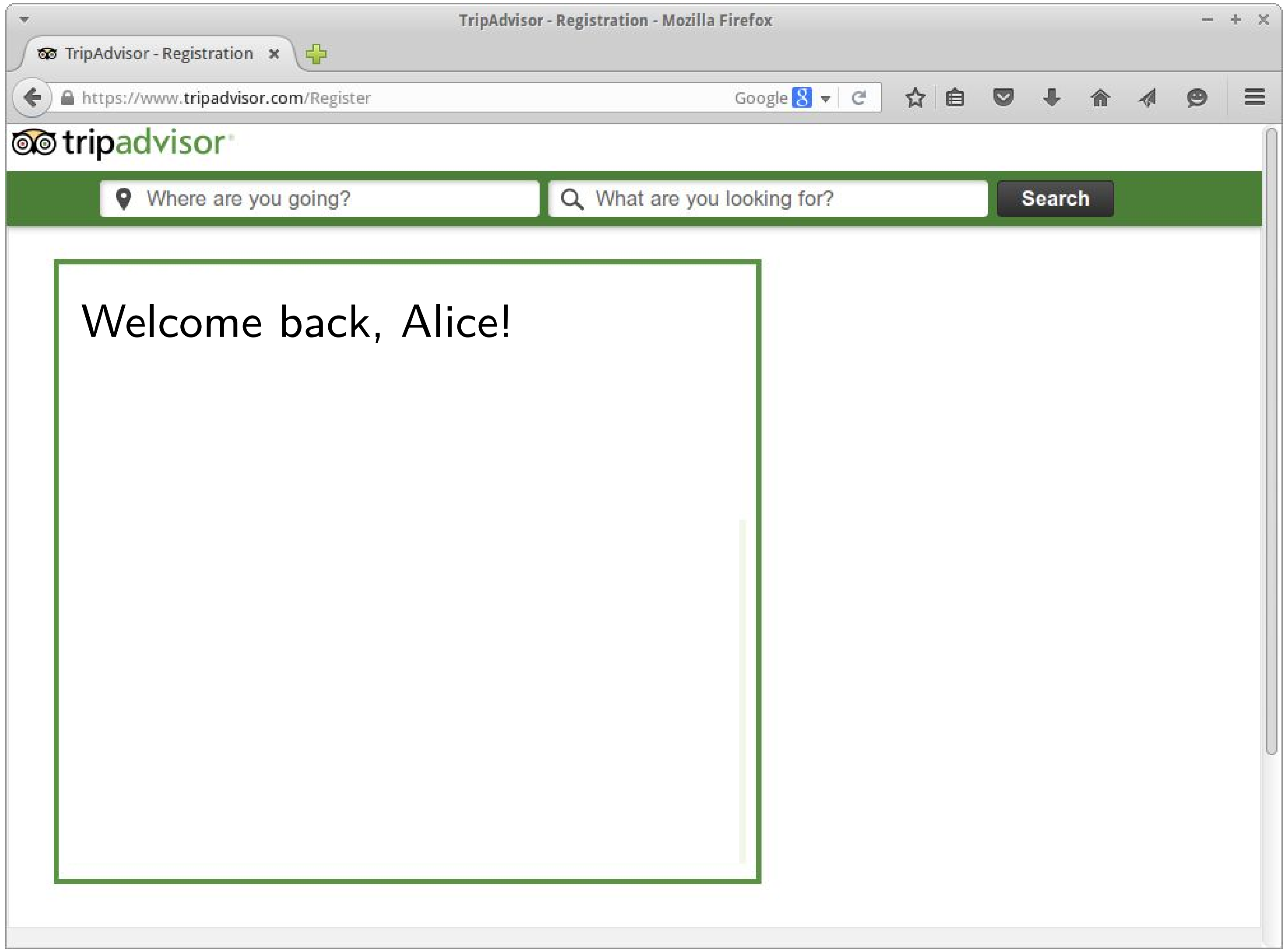
You can also log in with your SAMSUNG account

Sign in

OAuth 2.0

OpenID Connect

Single Sign-On (SSO): Social Web SSO – The Beginning


TripAdvisor - Registration - Mozilla Firefox
TripAdvisor - Registration
https://www.tripadvisor.com/Register
Google
tripadvisor
Where are you going?
What are you looking for?
Search

Welcome back, Alice!

OAuth 2.0

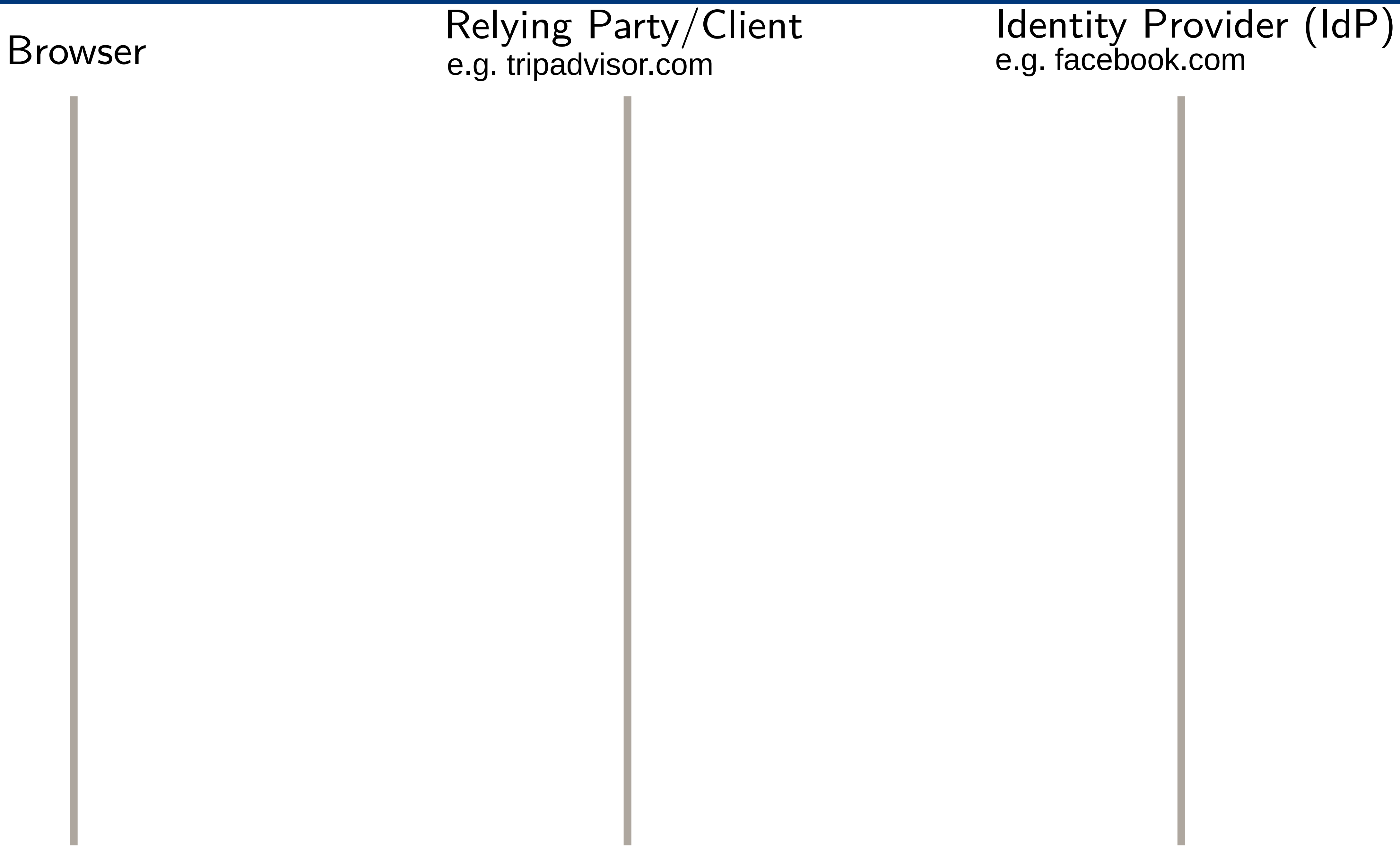
OpenID Connect

Single Sign-On (SSO): Social Web SSO – The Beginning

OAuth 2.0

OpenID Connect

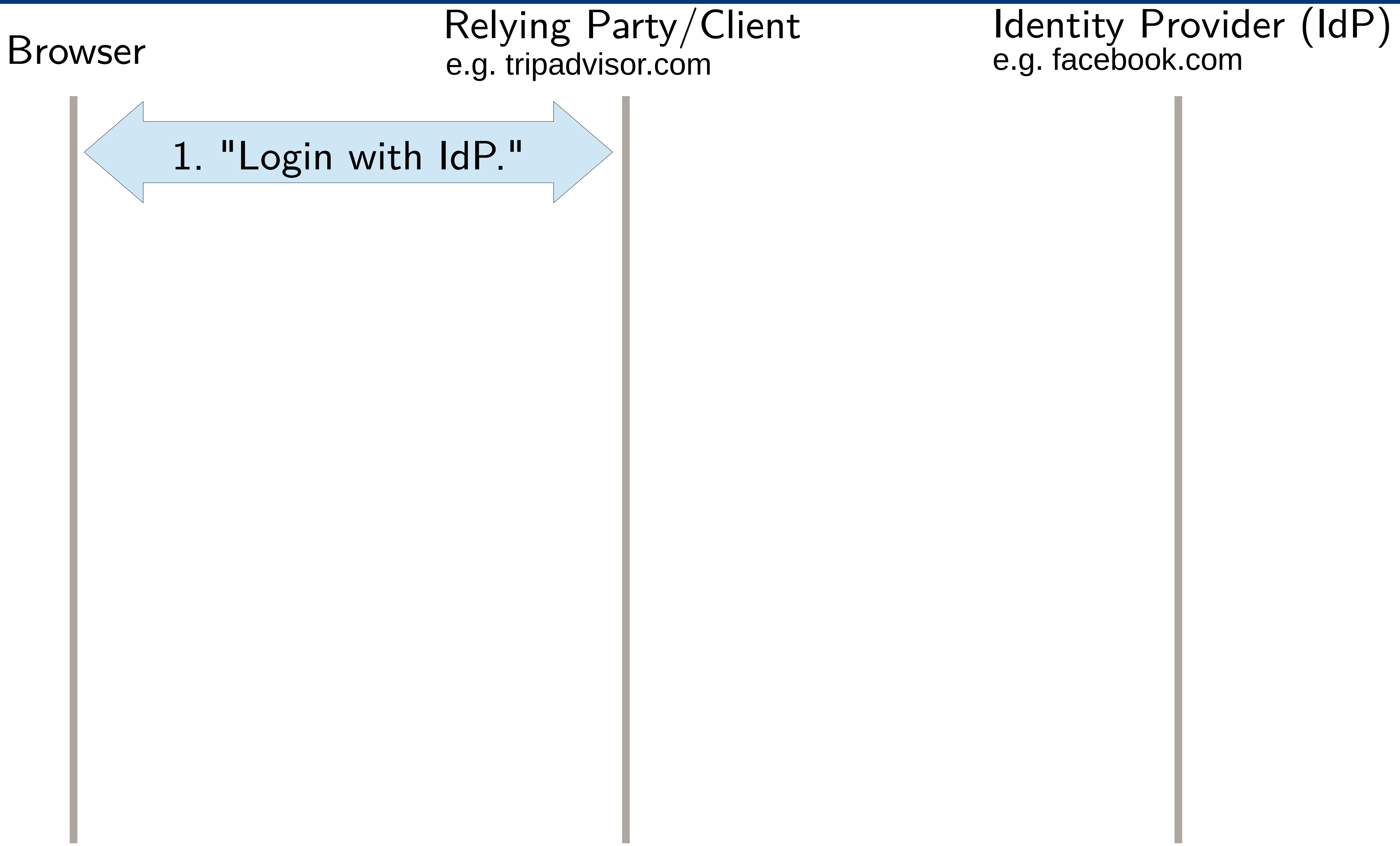
(Web) SSO: Basic Principle



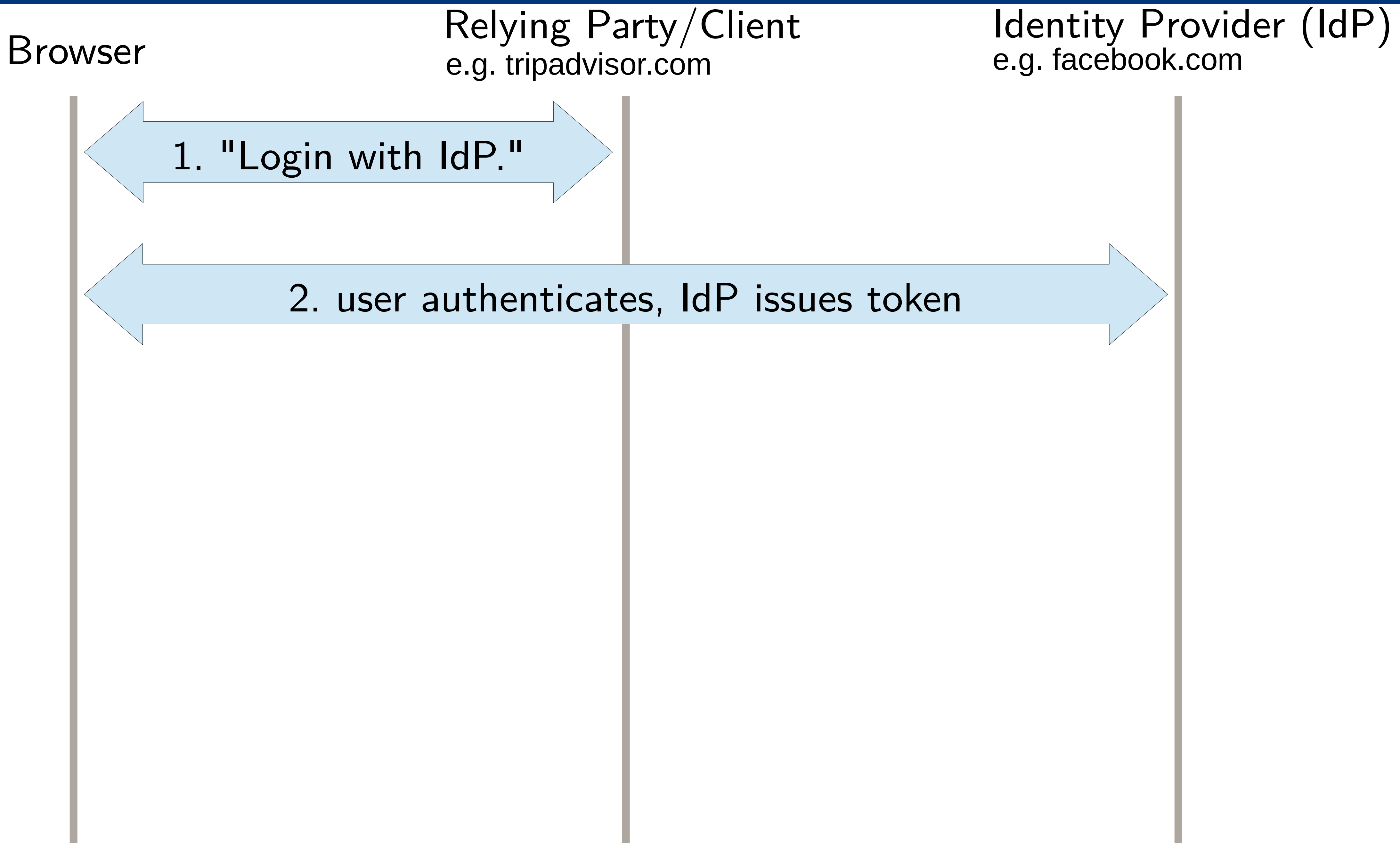
(Web) SSO: Basic Principle

Browser
Relying Party/Client
e.g. tripadvisor.com
Identity Provider (IdP)
e.g. facebook.com

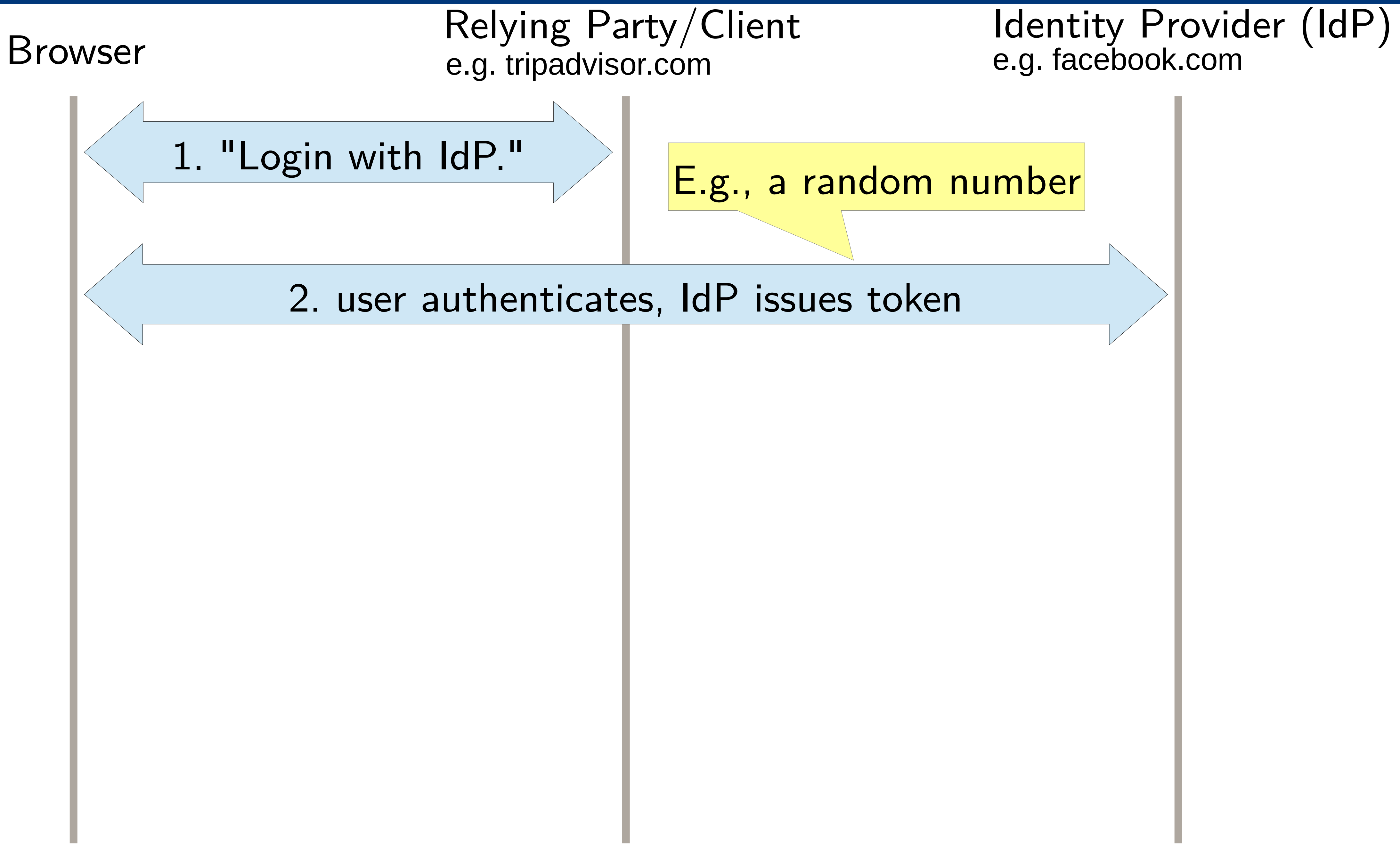
(Web) SSO: Basic Principle



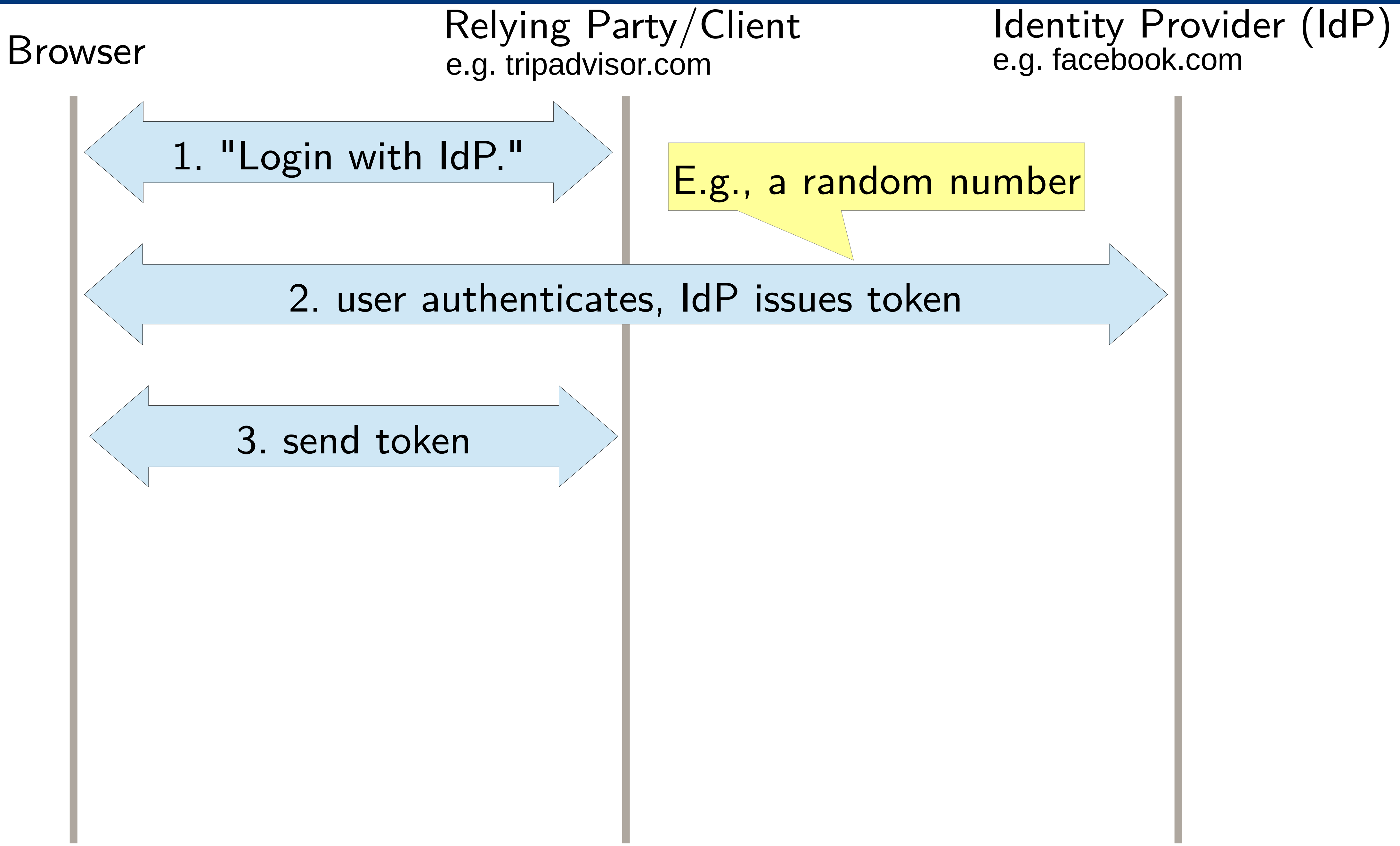
(Web) SSO: Basic Principle



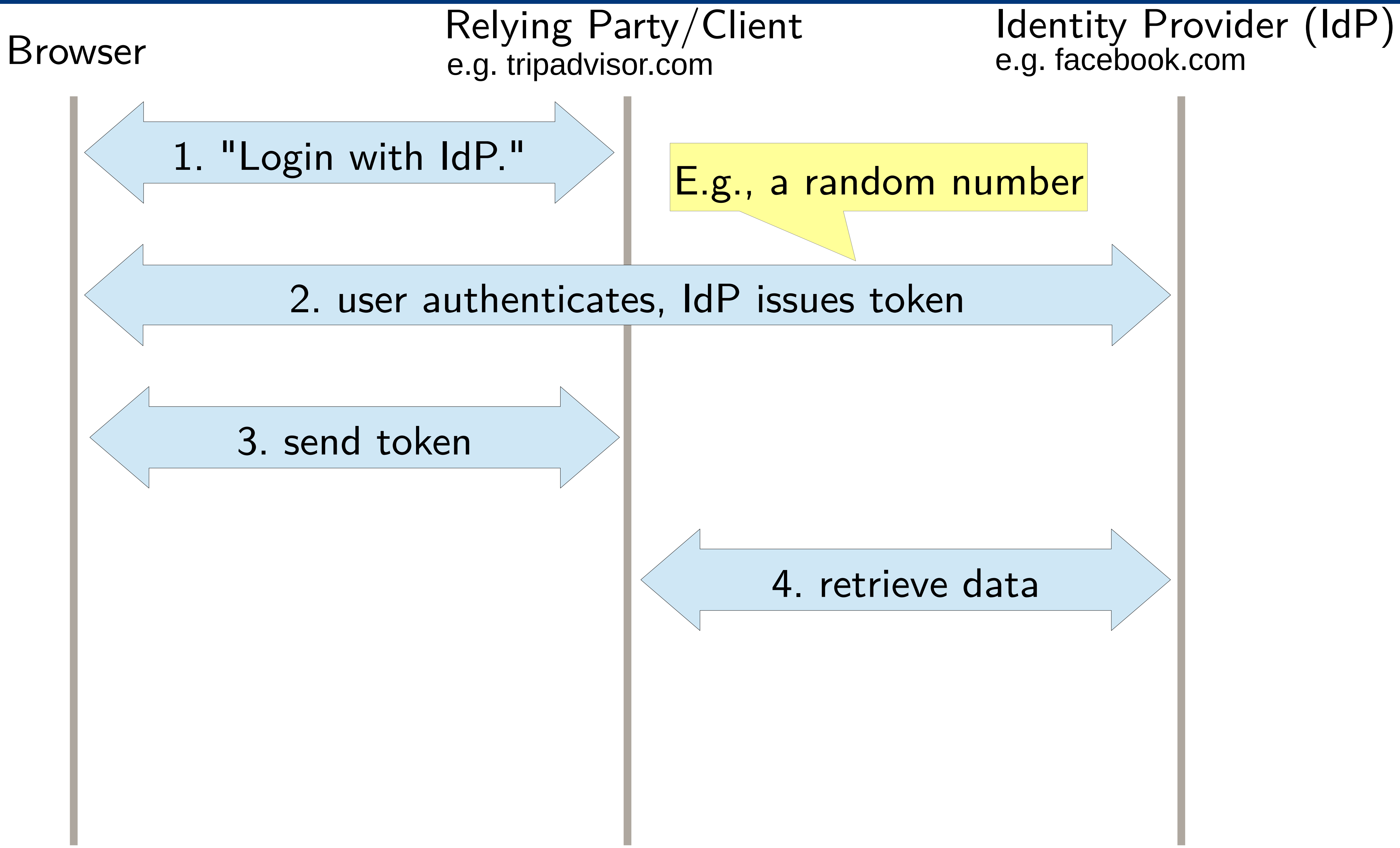
(Web) SSO: Basic Principle



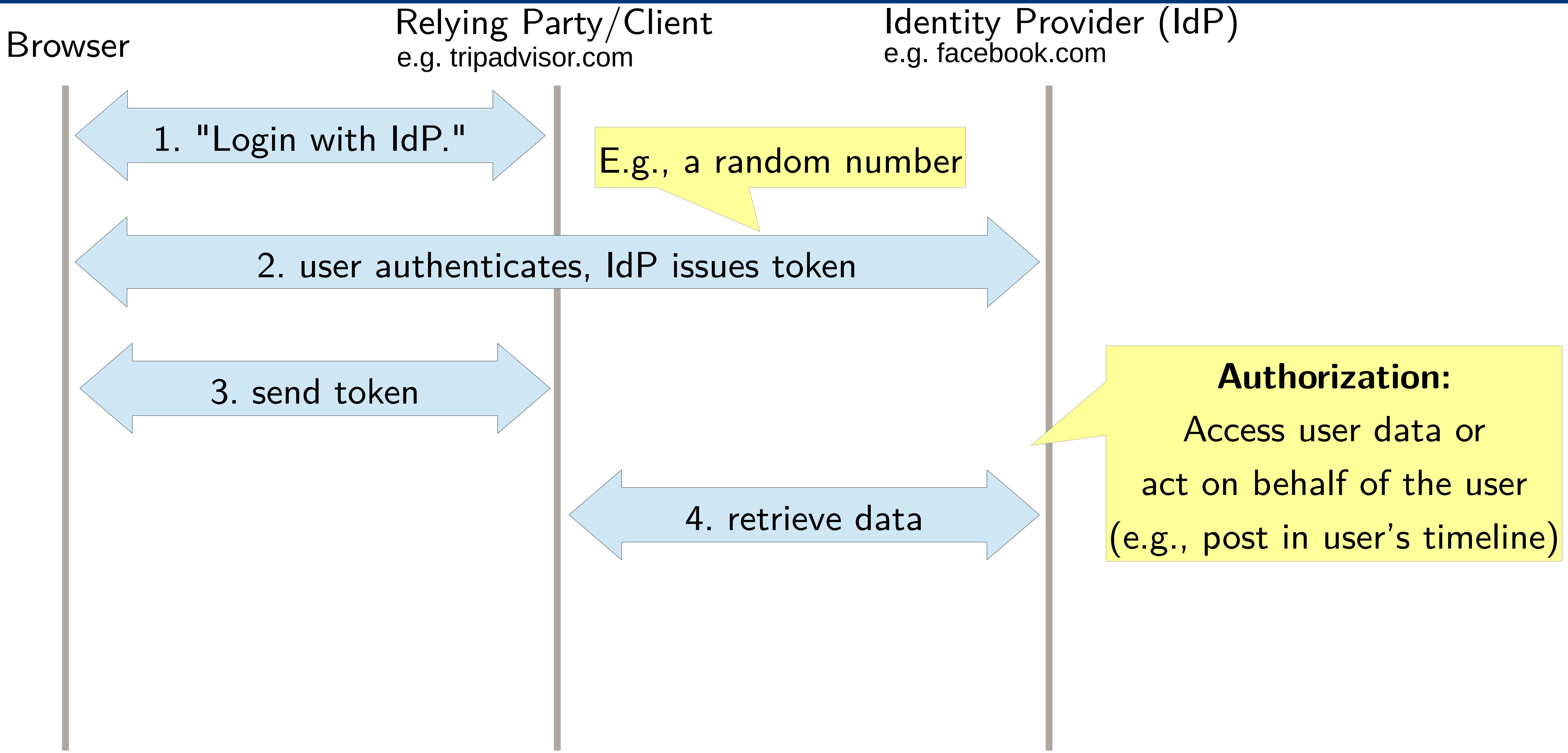
(Web) SSO: Basic Principle



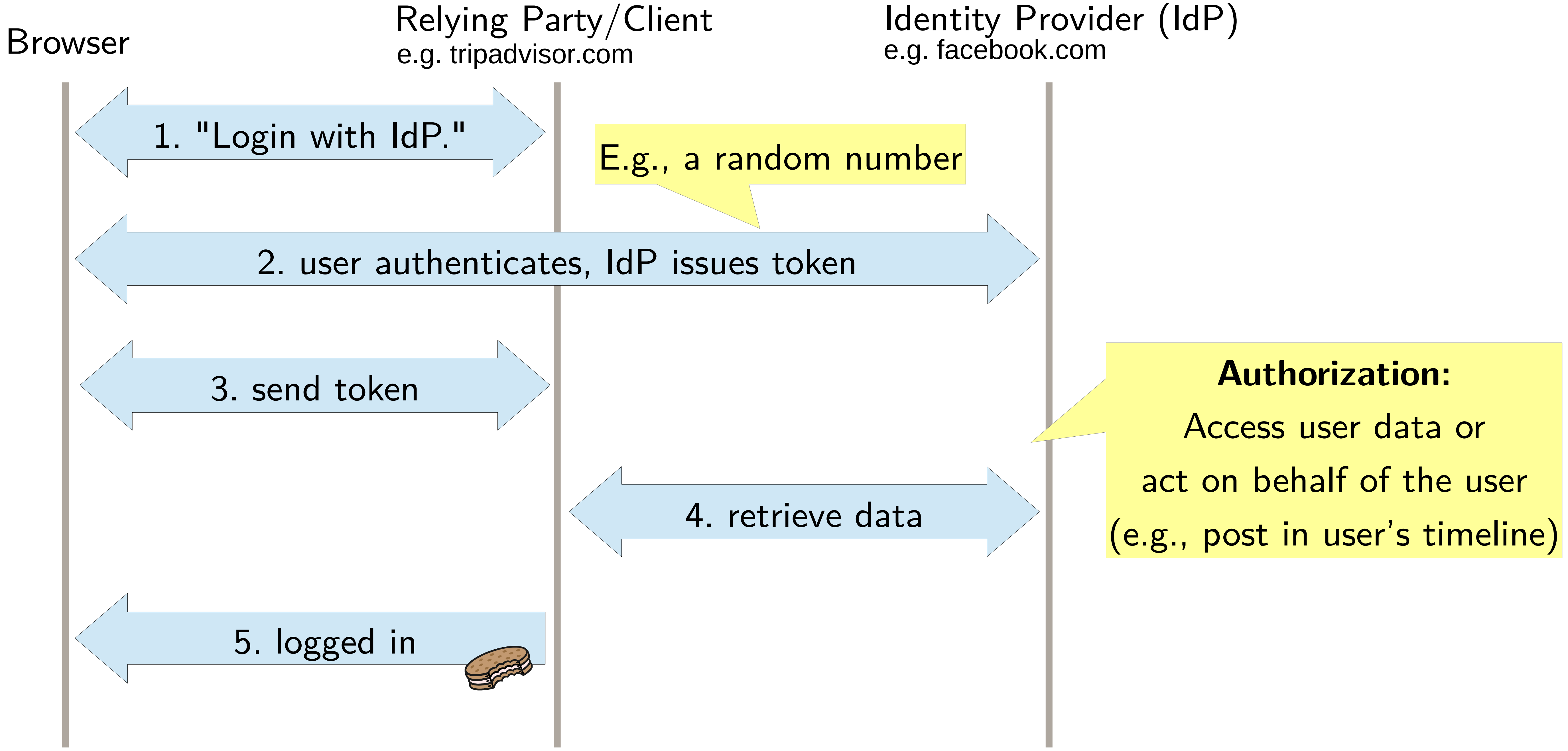
(Web) SSO: Basic Principle



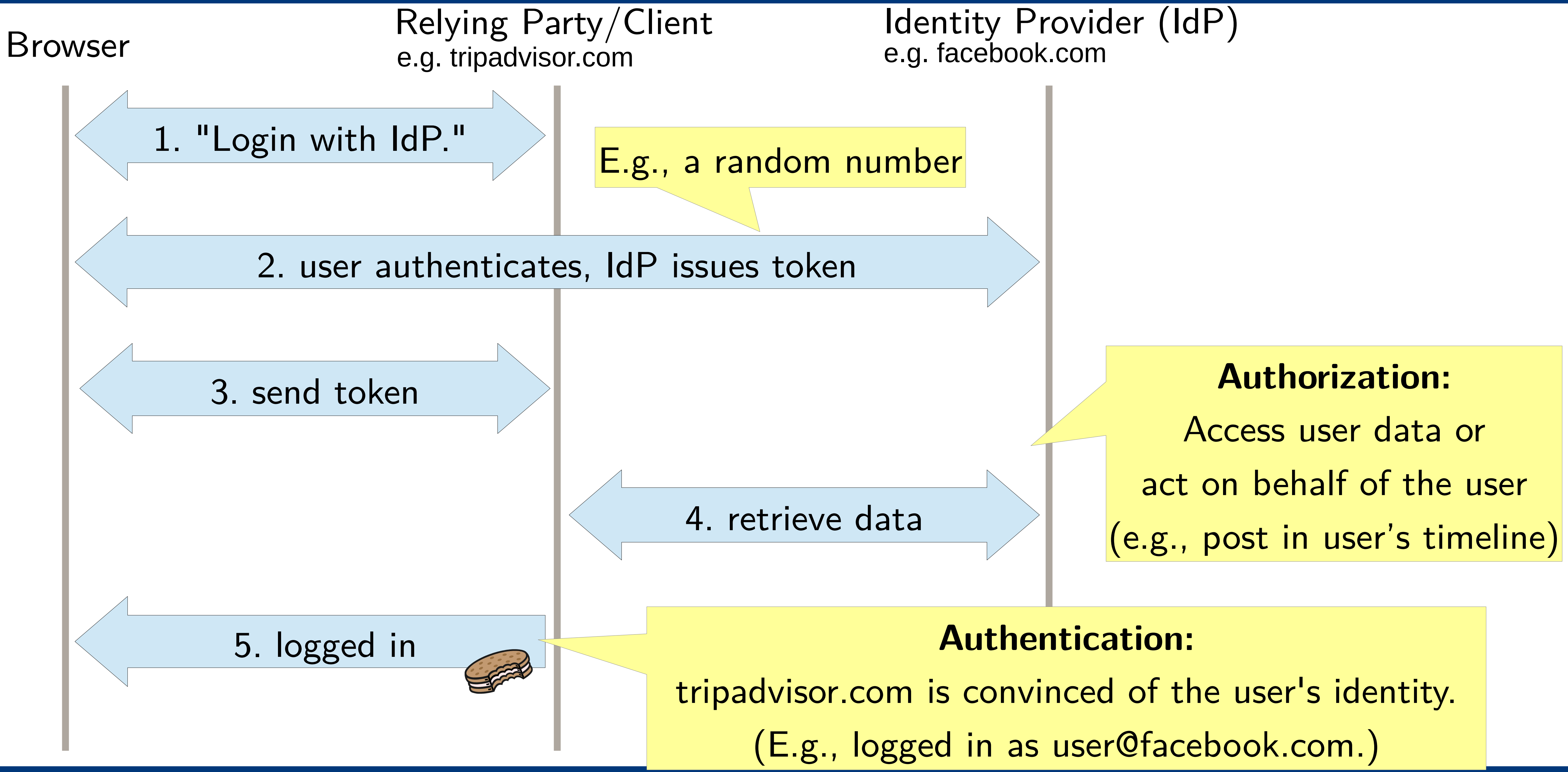
(Web) SSO: Basic Principle



(Web) SSO: Basic Principle



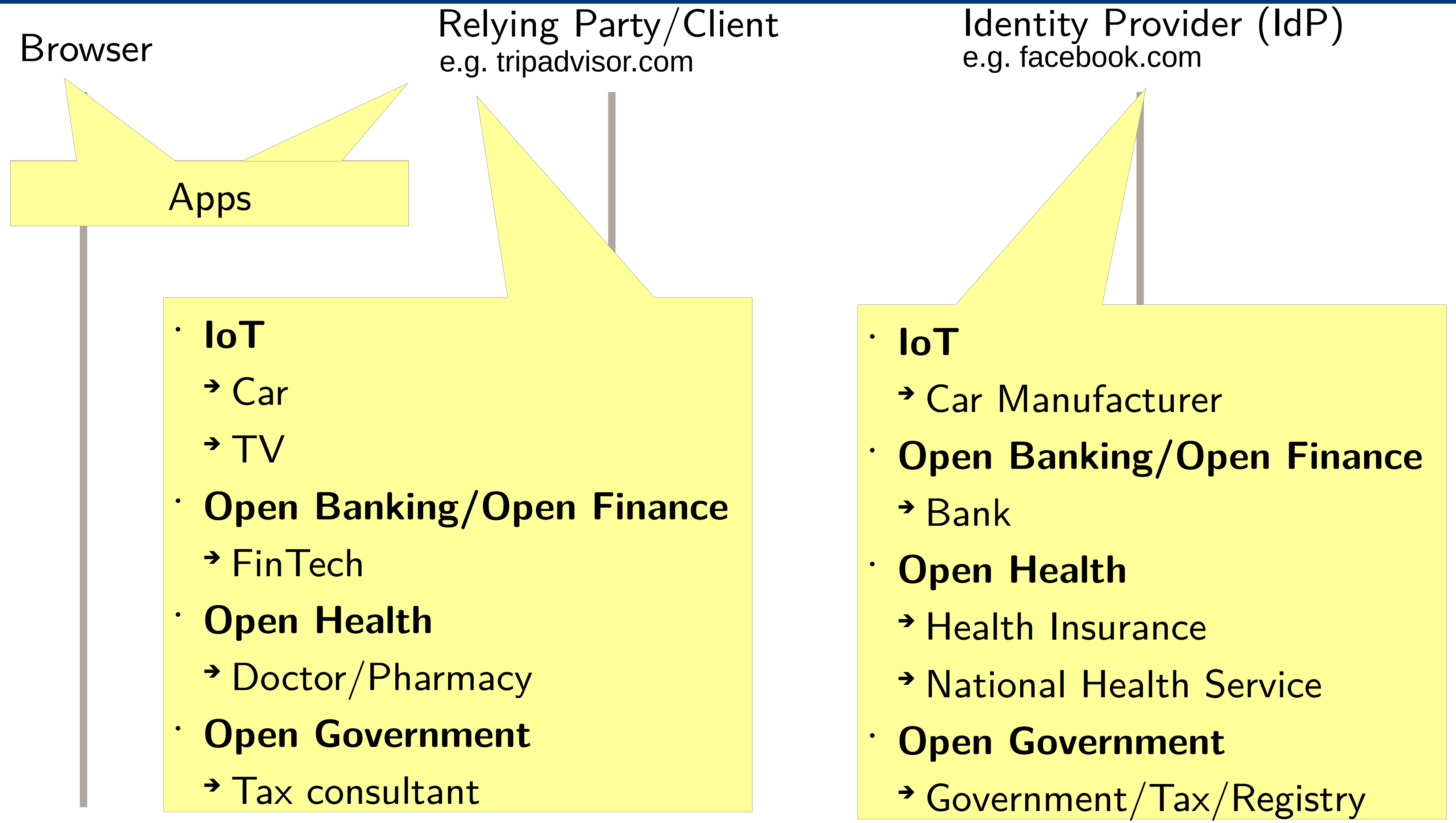
(Web) SSO: Basic Principle



Authorization:
Access user data or
act on behalf of the user
(e.g., post in user's timeline)

Authentication:
tripadvisor.com is convinced of the user's identity.
(E.g., logged in as user@facebook.com.)

SSO: Today



- **IoT**
  - Car
  - TV
- **Open Banking/Open Finance**
  - FinTech
- **Open Health**
  - Doctor/Pharmacy
- **Open Government**
  - Tax consultant

- **IoT**
  - Car Manufacturer
- **Open Banking/Open Finance**
  - Bank
- **Open Health**
  - Health Insurance
  - National Health Service
- **Open Government**
  - Government/Tax/Registry

SSO: Today

Browser

Relying Party/Client
e.g. tripadvisor.com

Identity Provider (IdP)
e.g. facebook.com

Apps

- **IoT**
  - Car
  - TV
- **Open Banking/Open Finance**
  - FinTech
- **Open Health**
  - Doctor/Pharmacy
- **Open Government**
  - Tax consultant

- **IoT**
  - Car Manufacturer
- **Open Banking/Open Finance**
  - Bank
- **Open Health**
  - Health Insurance
  - National Health Service
- **Open Government**
  - Government/Tax/Registry

SSO: Today

Browser

Relying Party/Client
e.g. tripadvisor.com

Identity Provider (IdP)
e.g. facebook.com

Apps

- **IoT**
  - Car
  - TV
- **Open Banking/Open Finance**
  - FinTech
- **Open Health**
  - Doctor/Pharmacy
- **Open Government**
  - Tax consultant

More secure and
complex protocols needed

Assume stronger and more
motivated attackers

acturer

- **Open Banking/Open Finance**
  - Bank
- **Open Health**
  - Health Insurance
  - National Health Service
- **Open Government**
  - Government/Tax/Registry

Our Goal About a Decade Ago

- A comprehensive model of the web infrastructure.
- To formally model and analyze web applications, protocols, and standards.

Result:

The Web Infrastructure Model (WIM) [S&P14]

- At that time only very limited models existed:
  - Kerschbaum as well as Akhawe et al. (Alloy models)
  - Bansal et al. (Proverif model)
- The WIM is still the by far most comprehensive model of the web infrastructure.

  [S&P14], [ESORICS15], [CCS15], [CCS16], [CSF17], [S&P19], [S&P22], [ESORICS23], [CSF24], [ACM TOPS24]

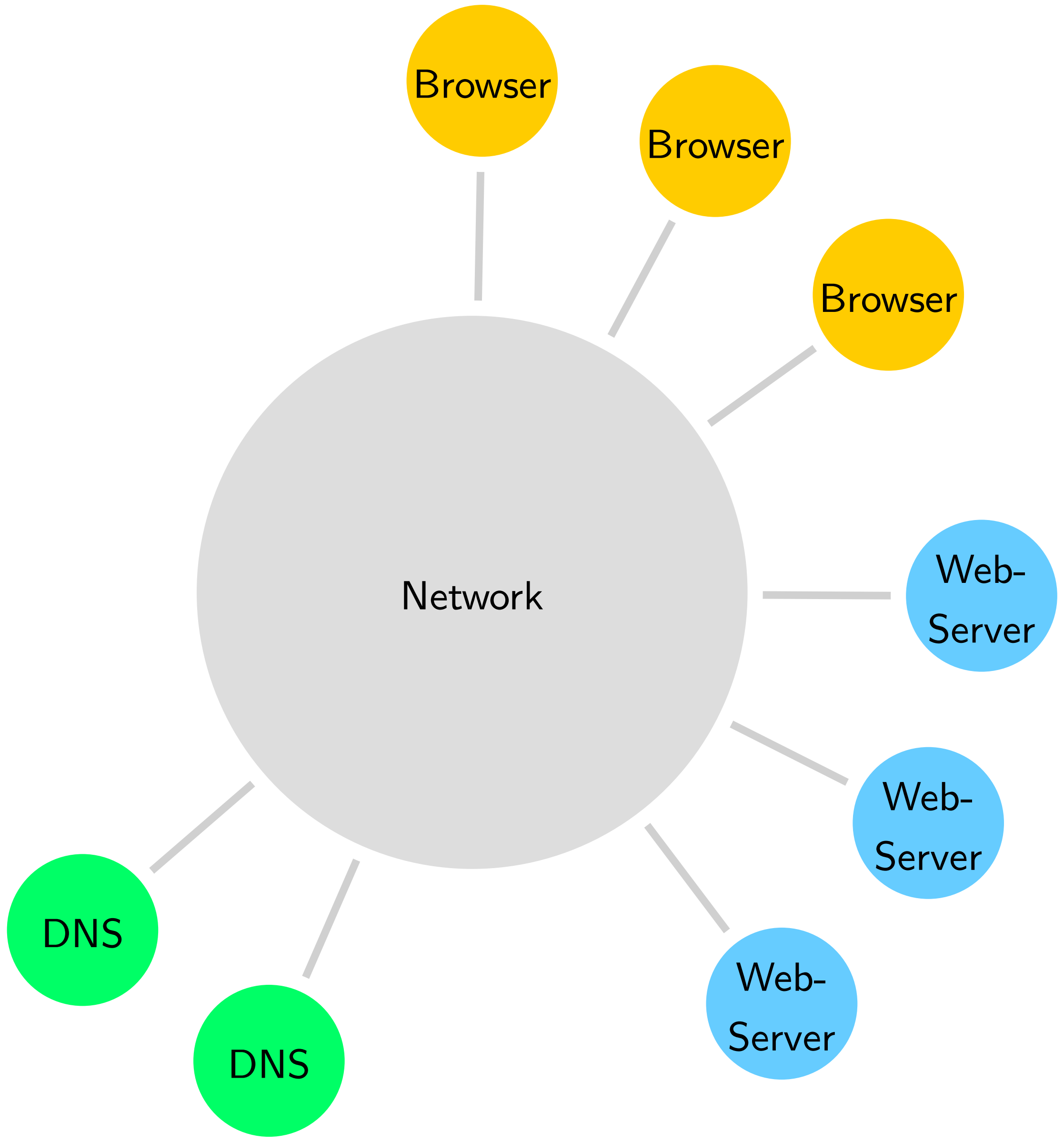
The Web Infrastructure Model (WIM)

Sources

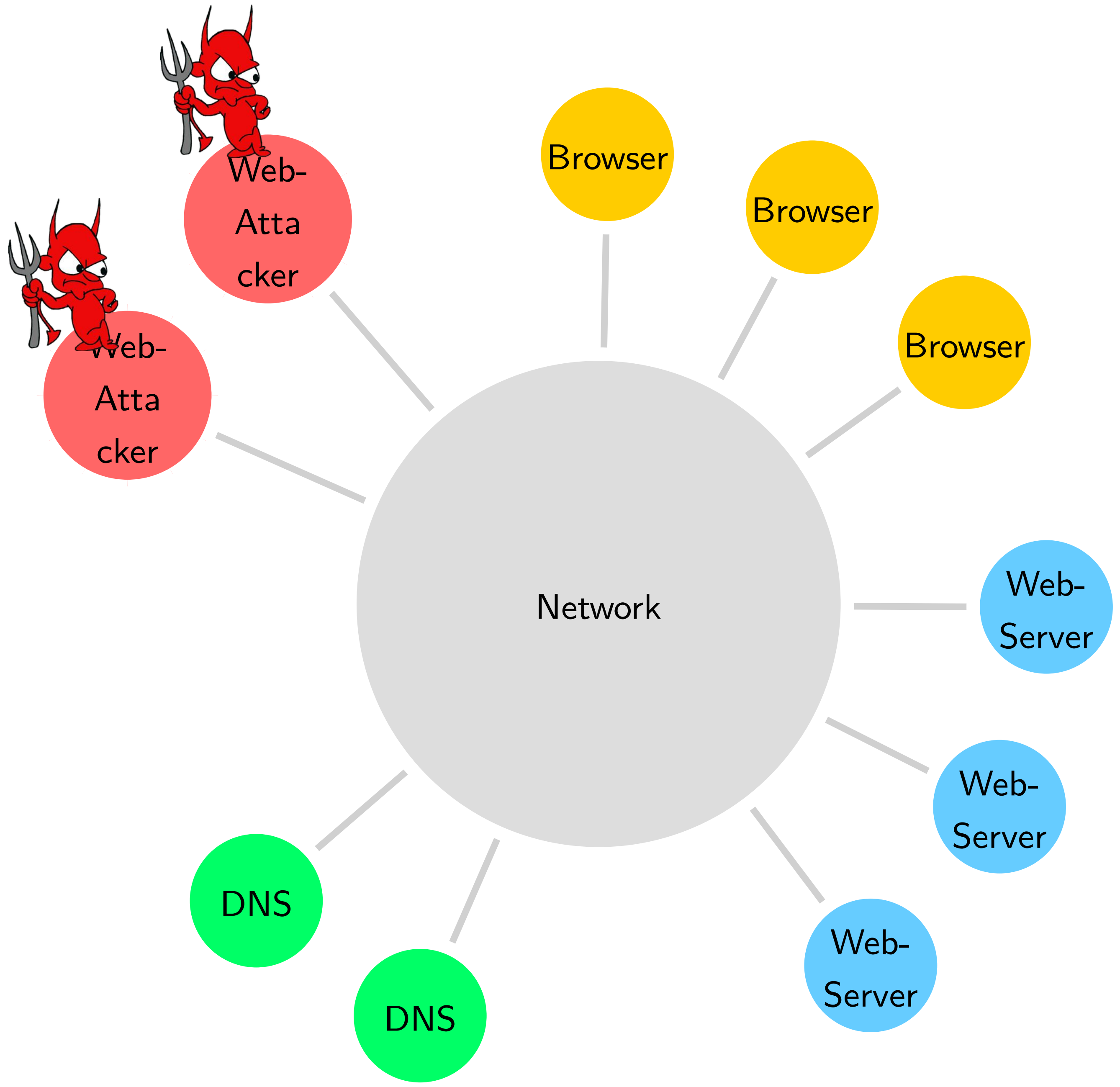
Specifications for the web are spread across many sources with mutual dependencies:

- Standards and RFCs
  - HTTP/1.1, HTTP/2, HTTP/3 Standards
  - W3C HTML5
  - W3C Web Storage
  - WHATWG Fetch
  - W3C Cross-Origin Resource Sharing
  - RFCs (6265, 6797, 6454, 2616, …)
- Browser implementations
  - Google Chrome
  - Mozilla Firefox
  - …

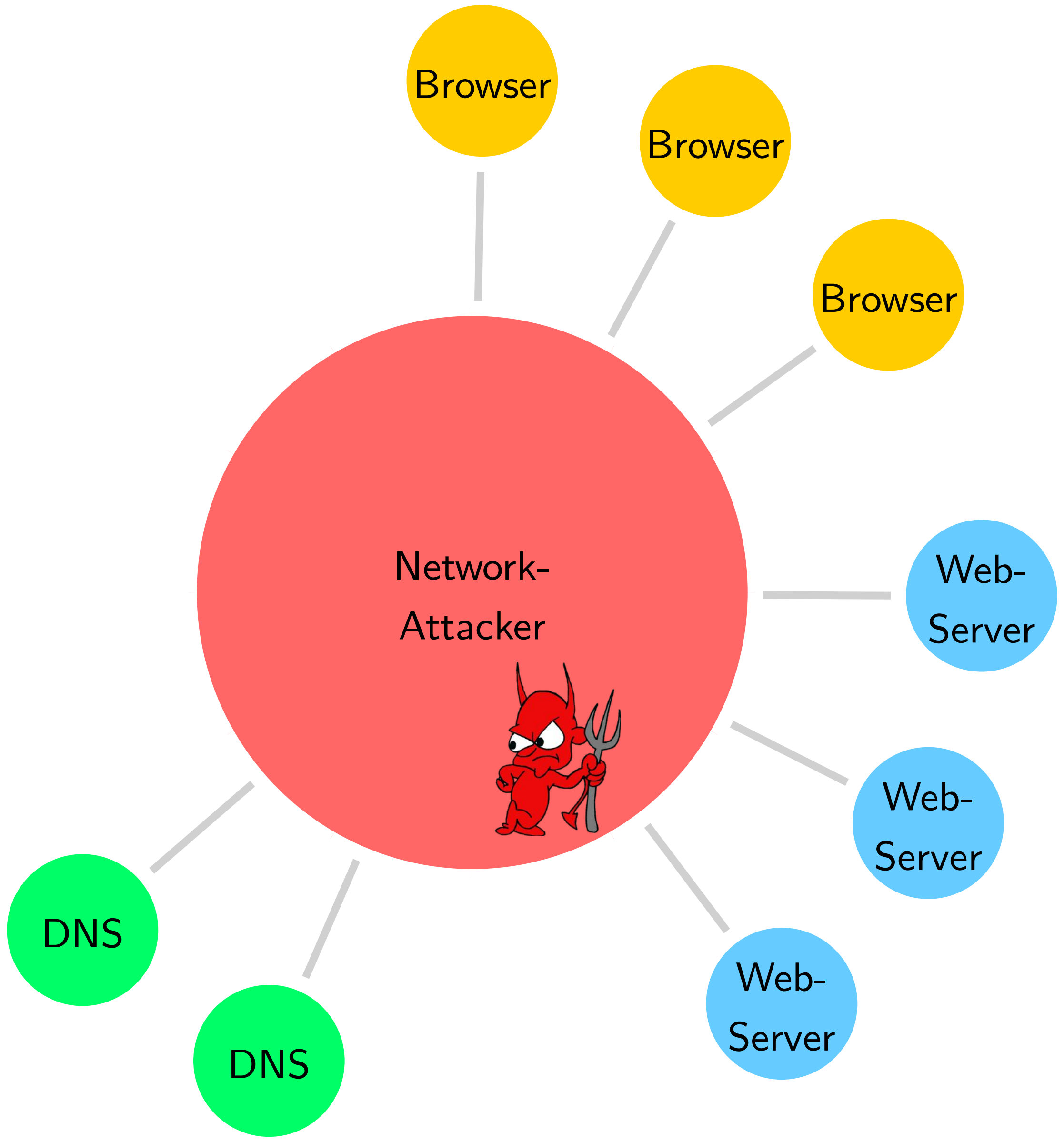
WIM: Network Model and Attackers



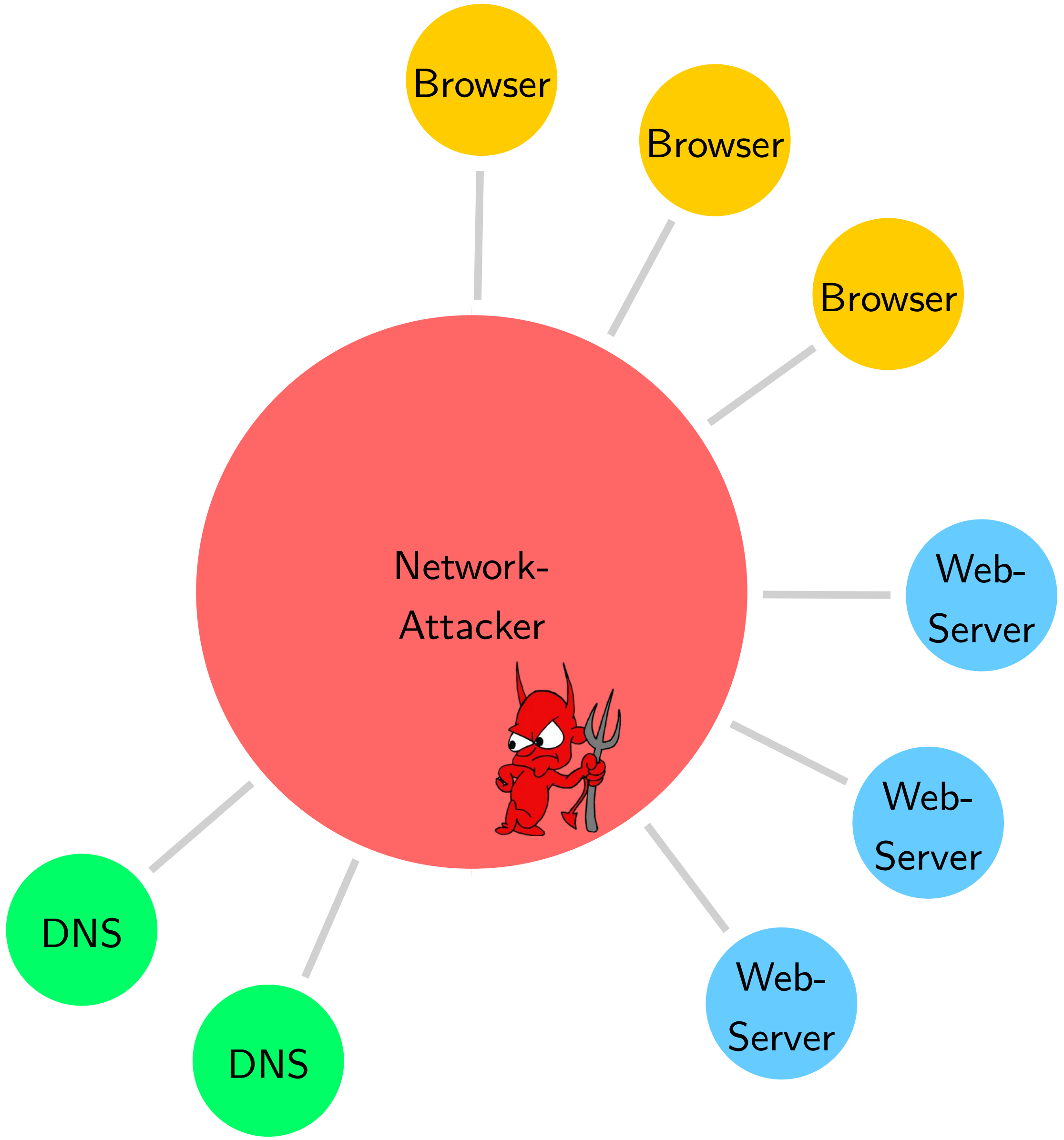
WIM: Network Model and Attackers



WIM: Network Model and Attackers

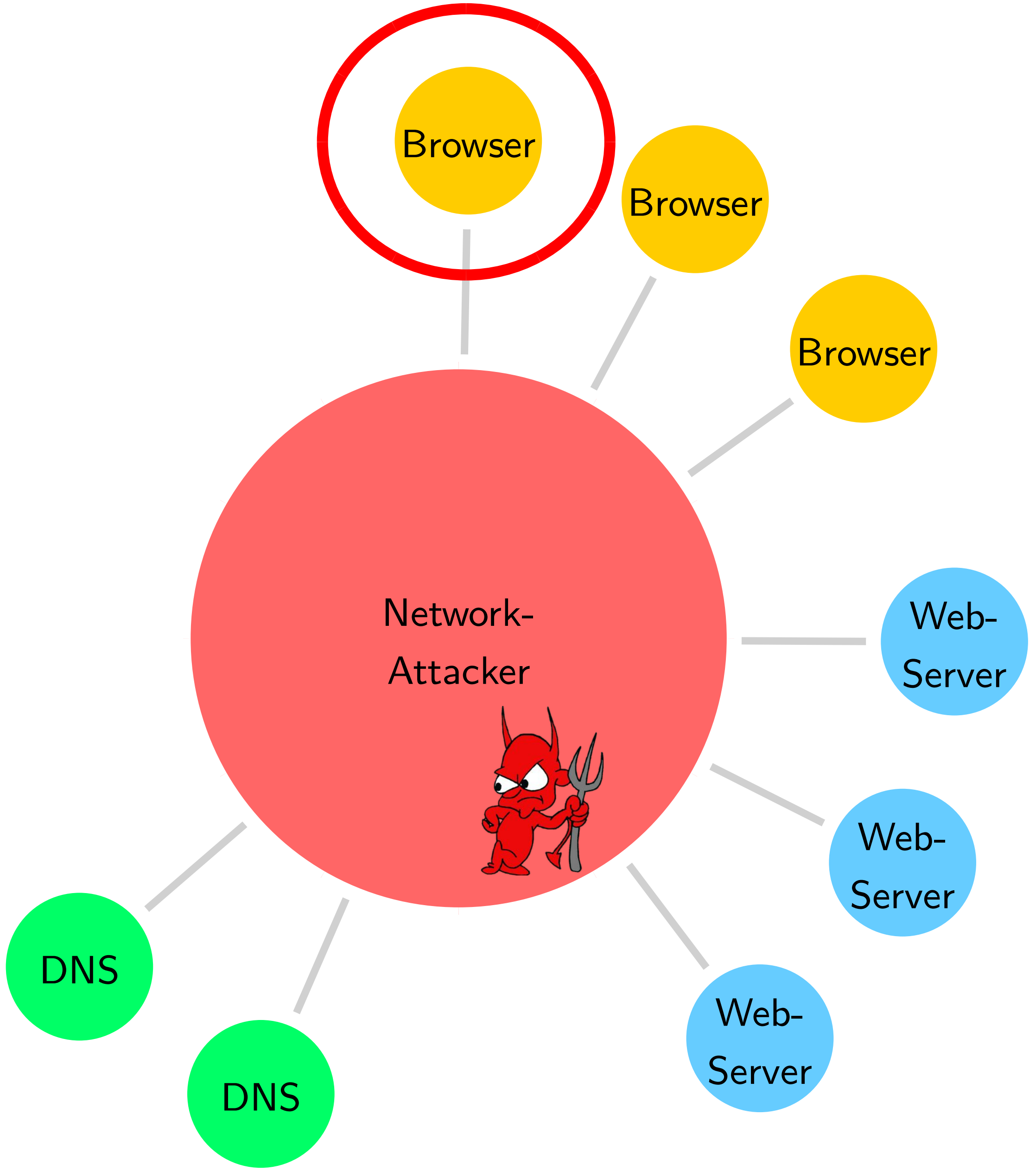


WIM: Network Model and Attackers



Dolev-Yao-Attacker

WIM: Network Model and Attackers



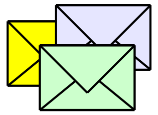
Dolev-Yao-Attacker

WIM: Web Browser Model

WIM: Web Browser Model

Including ...

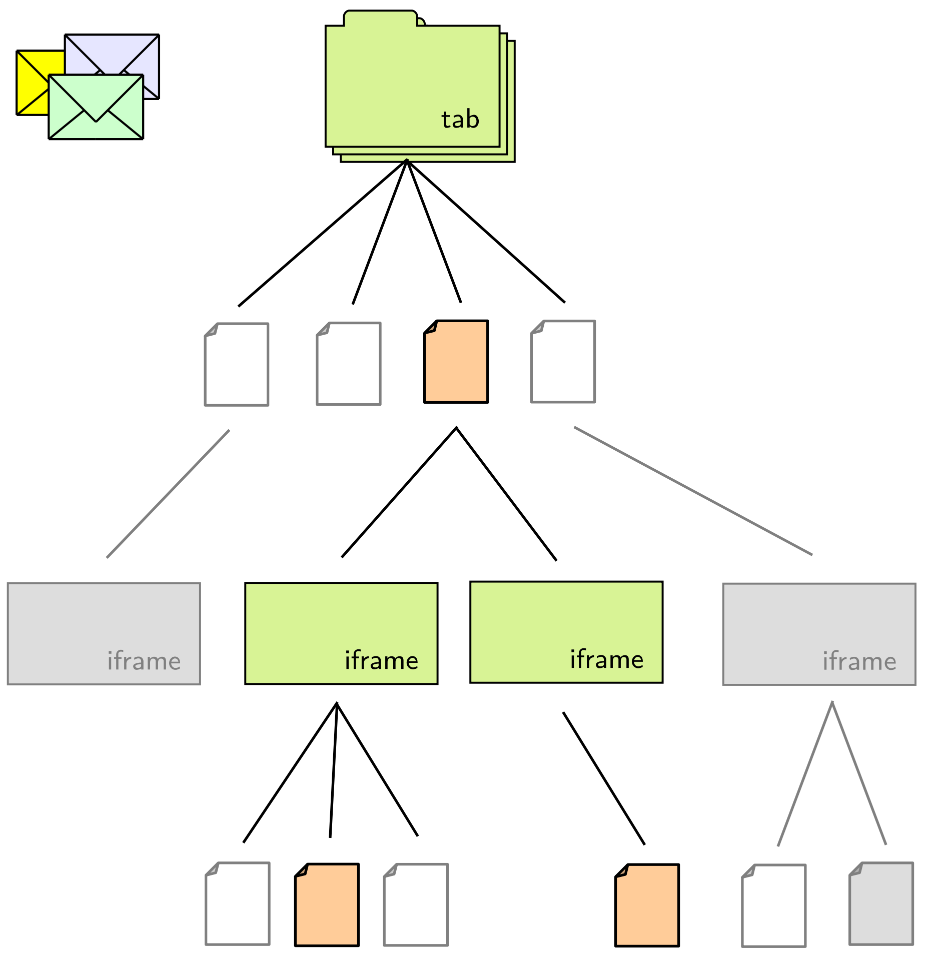
WIM: Web Browser Model



Including …

- DNS, HTTP, HTTPS

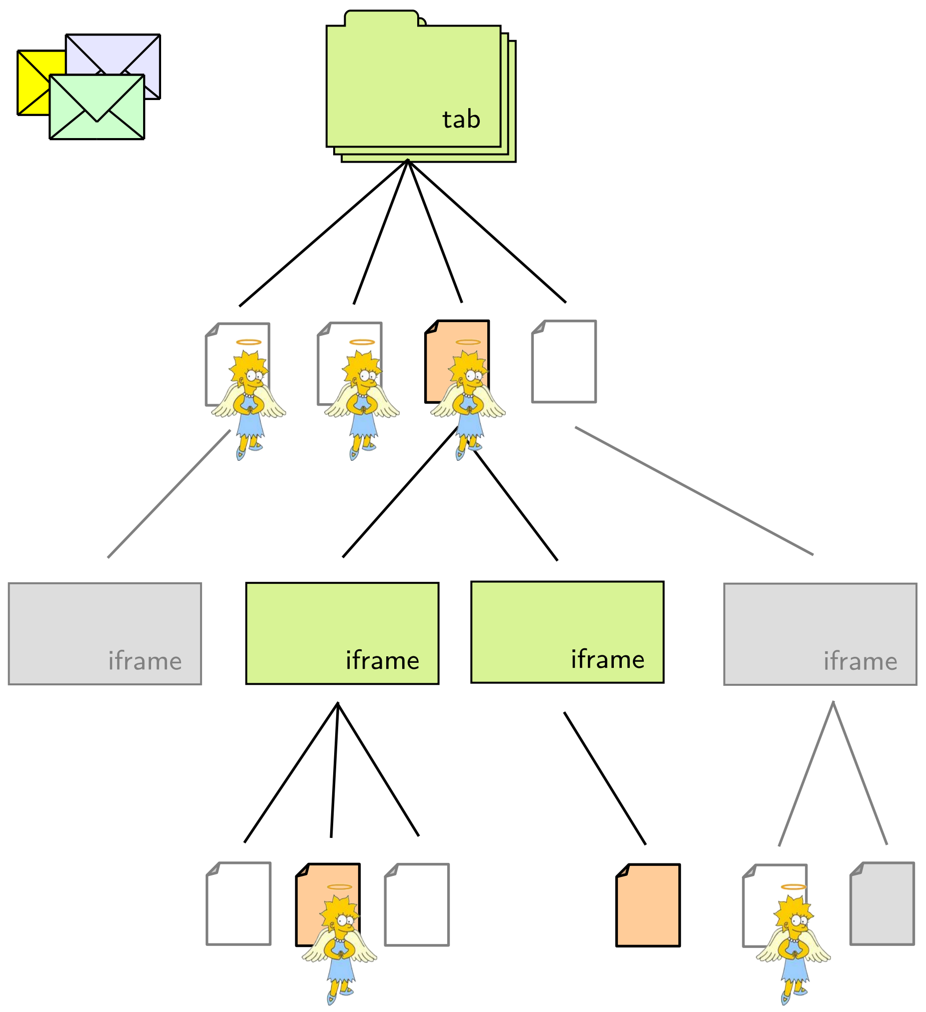
WIM: Web Browser Model



Including ...

- DNS, HTTP, HTTPS
- window & document structure

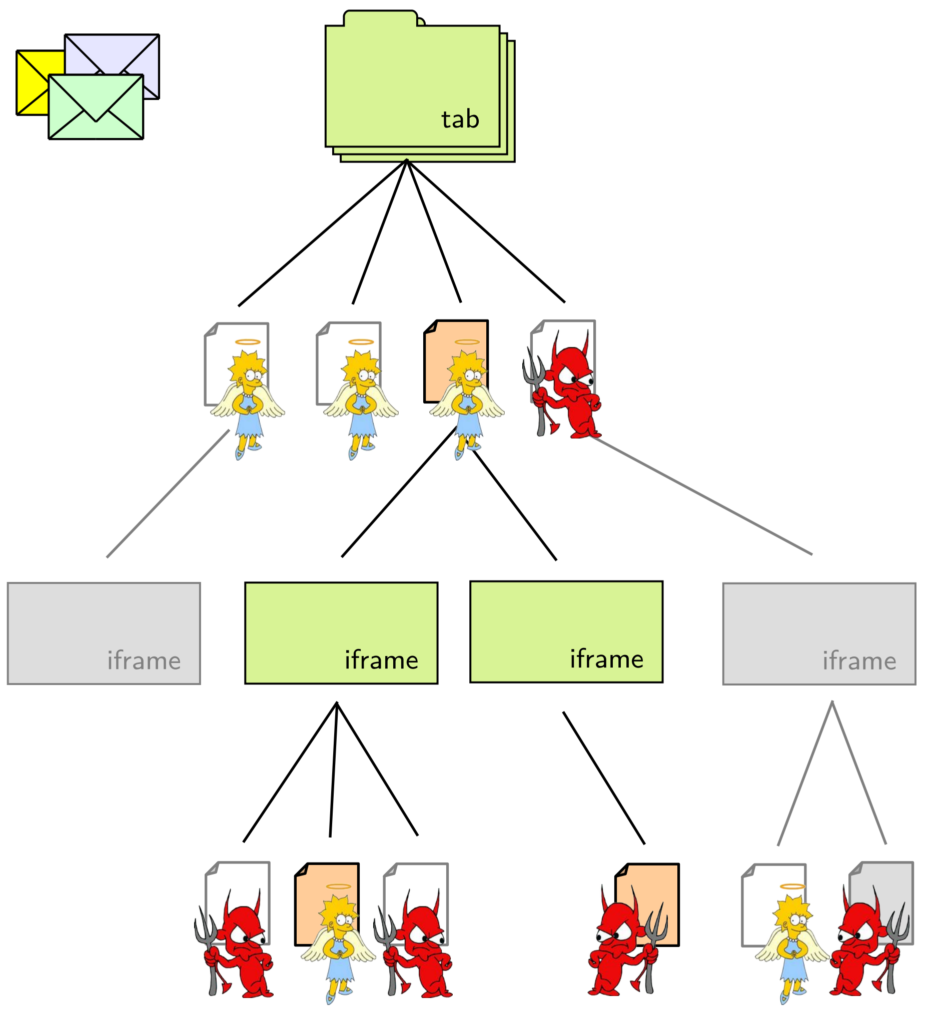
WIM: Web Browser Model



Including …

- DNS, HTTP, HTTPS
- window & document structure
- scripts

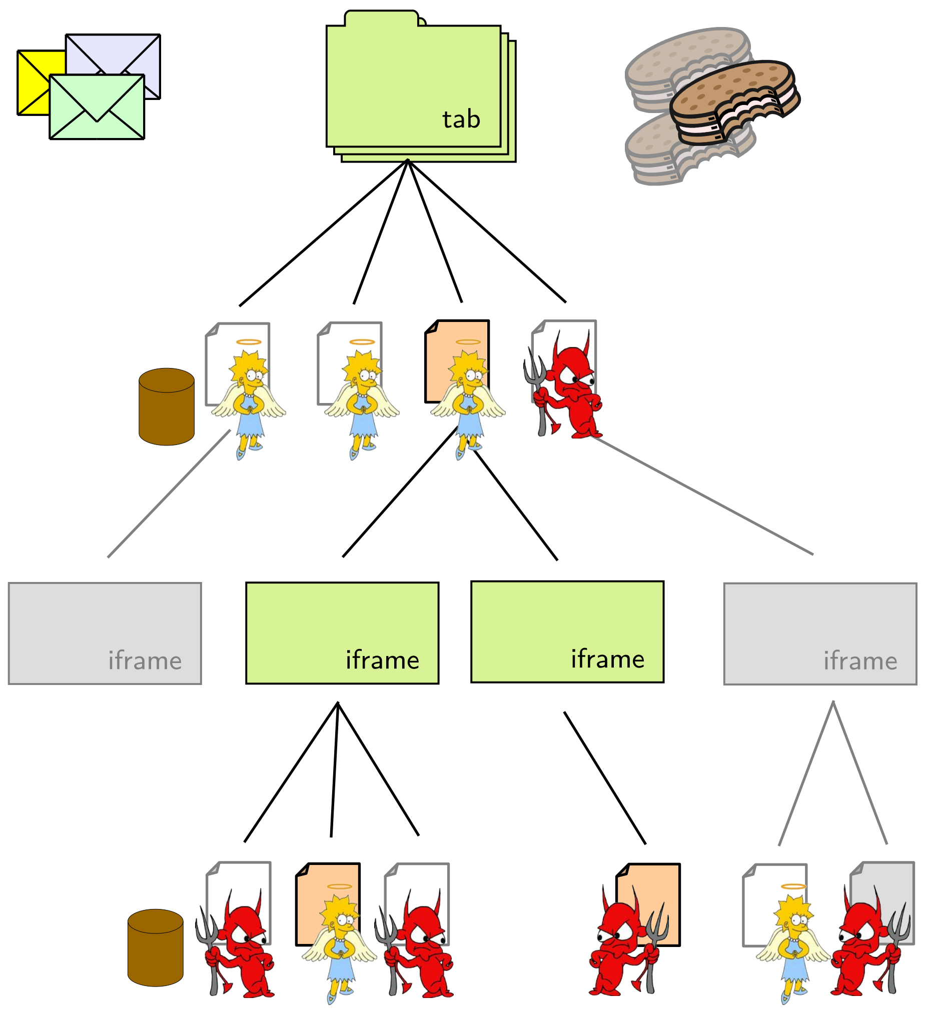
WIM: Web Browser Model



Including ...

- DNS, HTTP, HTTPS
- window & document structure
- scripts
- attacker scripts

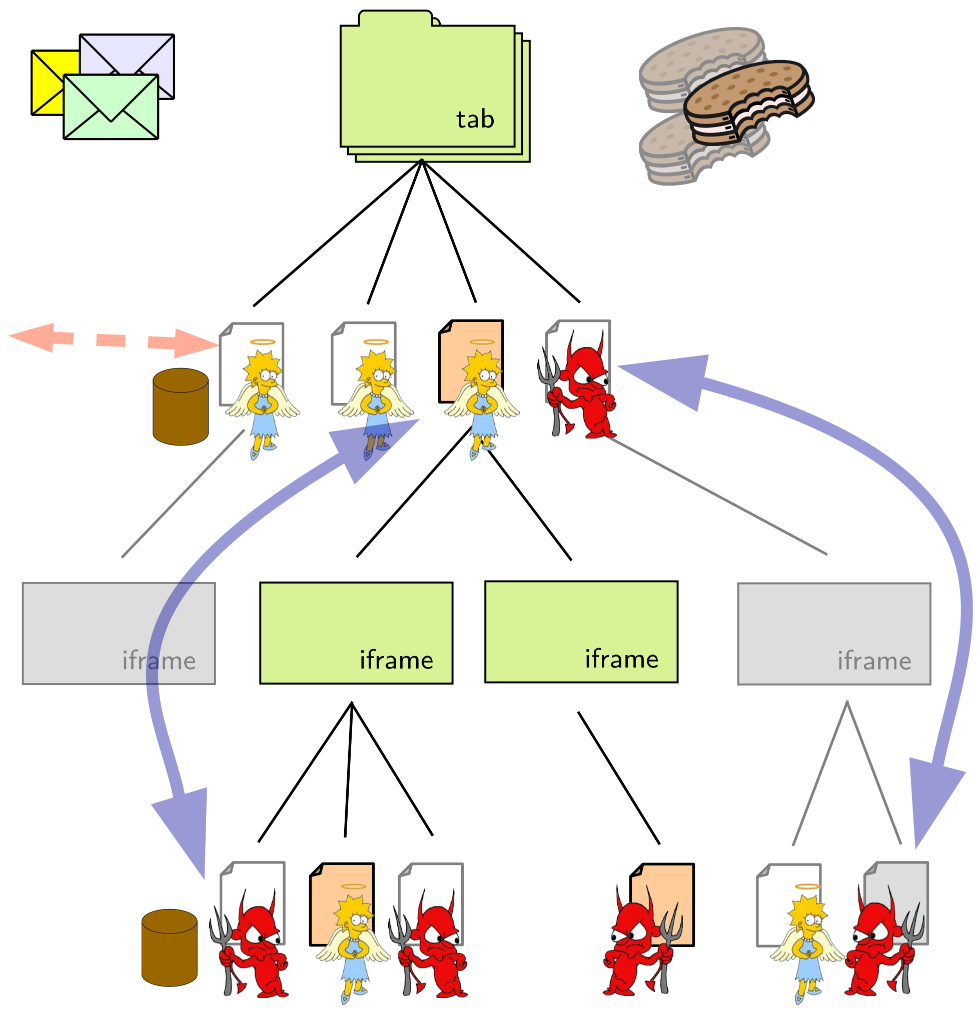
WIM: Web Browser Model



Including ...

- DNS, HTTP, HTTPS
- window & document structure
- scripts
- attacker scripts
- web storage & cookies

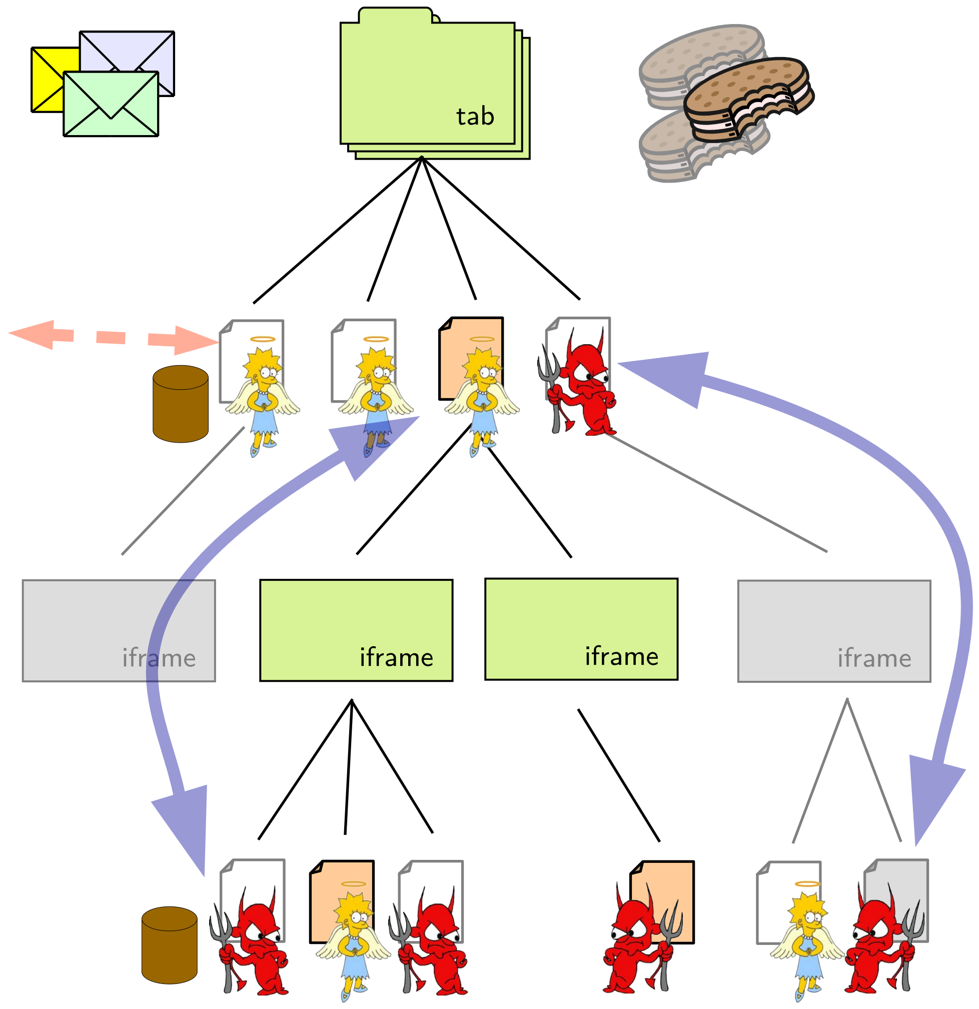
WIM: Web Browser Model



Including ...

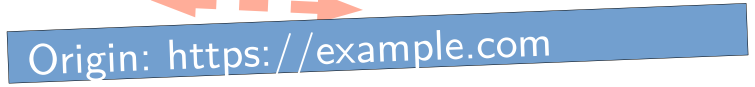
- DNS, HTTP, HTTPS
- window & document structure
- scripts
- attacker scripts
- web storage & cookies
- web messaging & XHR

WIM: Web Browser Model

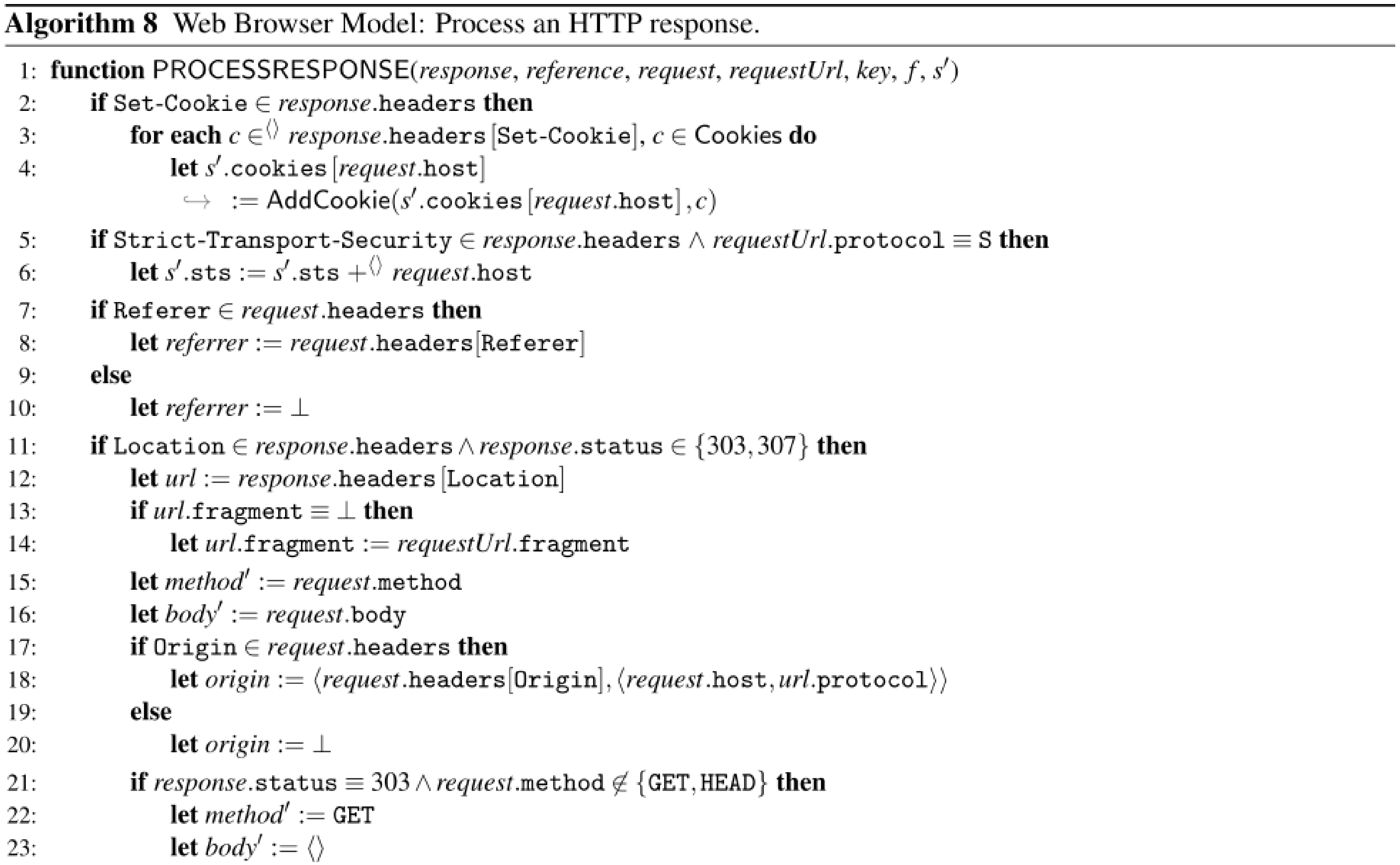


Including ...

- DNS, HTTP, HTTPS
- window & document structure
- scripts
- attacker scripts
- web storage & cookies
- web messaging & XHR
- message headers
- redirections
- security policies
- dynamic corruption
- WebRTC
- ...



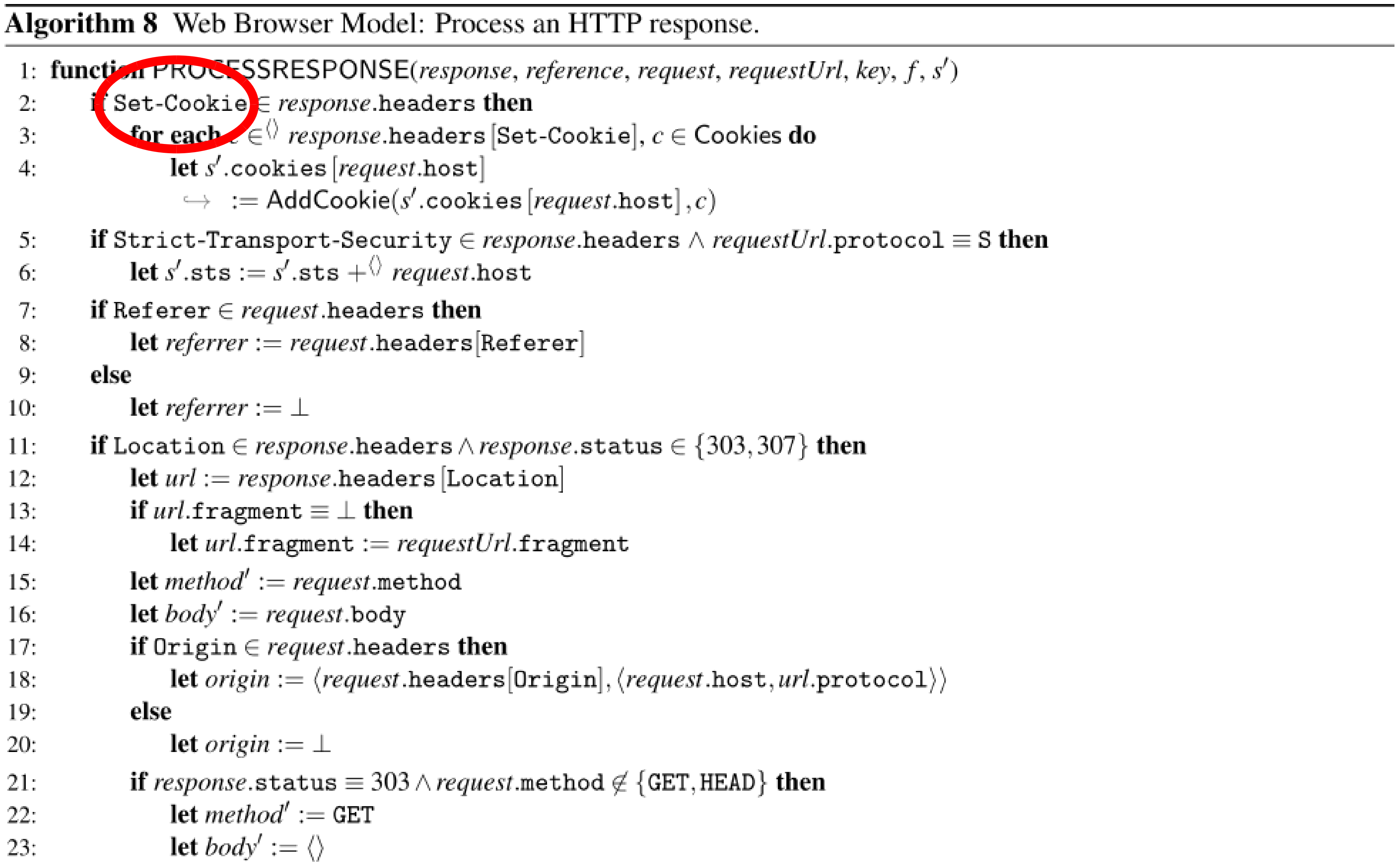
WIM: Web Browser Model – Excerpt



Algorithm 8 Web Browser Model: Process an HTTP response.

```
1:  function PROCESSRESPONSE($response, reference, request, requestUrl, key, f, s'$)
2:      if Set-Cookie $\in response$.headers then
3:          for each $c \in^{\langle\rangle} response$.headers[Set-Cookie], $c \in$ Cookies do
4:              let $s'$.cookies[$request$.host]
                  $\hookrightarrow$ := AddCookie($s'$.cookies[$request$.host], $c$)
5:      if Strict-Transport-Security $\in response$.headers $\wedge$ $requestUrl$.protocol $\equiv$ S then
6:          let $s'$.sts := $s'$.sts $+^{\langle\rangle}$ $request$.host
7:      if Referer $\in request$.headers then
8:          let $referrer$ := $request$.headers[Referer]
9:      else
10:         let $referrer := \perp$
11:     if Location $\in response$.headers $\wedge$ $response$.status $\in \{303, 307\}$ then
12:         let $url$ := $response$.headers[Location]
13:         if $url$.fragment $\equiv \perp$ then
14:             let $url$.fragment := $requestUrl$.fragment
15:         let $method'$ := $request$.method
16:         let $body'$ := $request$.body
17:         if Origin $\in request$.headers then
18:             let $origin$ := $\langle request$.headers[Origin], $\langle request$.host, $url$.protocol$\rangle\rangle$
19:         else
20:             let $origin := \perp$
21:         if $response$.status $\equiv 303 \wedge request$.method $\notin$ {GET, HEAD} then
22:             let $method'$ := GET
23:             let $body'$ := $\langle\rangle$
```

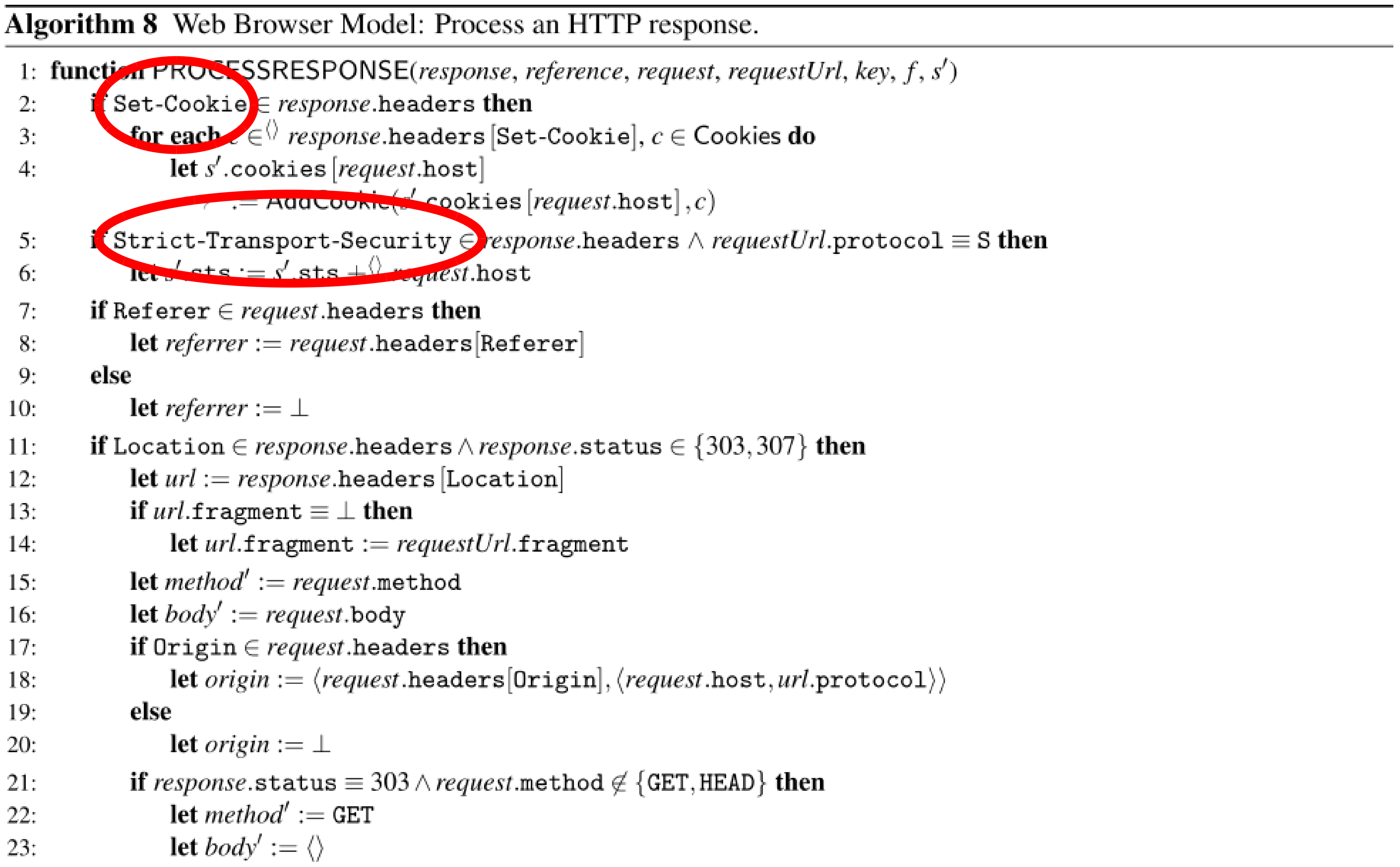
WIM: Web Browser Model – Excerpt



Algorithm 8 Web Browser Model: Process an HTTP response.

```
function PROCESSRESPONSE($response, reference, request, requestUrl, key, f, s'$)
    if Set-Cookie $\in$ $response$.headers then
        for each $c \in^{\langle\rangle}$ $response$.headers[Set-Cookie], $c \in$ Cookies do
            let $s'$.cookies[$request$.host]
              $\hookrightarrow$ := AddCookie($s'$.cookies[$request$.host], $c$)
    if Strict-Transport-Security $\in$ $response$.headers $\wedge$ $requestUrl$.protocol $\equiv$ S then
        let $s'$.sts := $s'$.sts $+^{\langle\rangle}$ $request$.host
    if Referer $\in$ $request$.headers then
        let $referrer$ := $request$.headers[Referer]
    else
        let $referrer$ := $\perp$
    if Location $\in$ $response$.headers $\wedge$ $response$.status $\in \{303, 307\}$ then
        let $url$ := $response$.headers[Location]
        if $url$.fragment $\equiv \perp$ then
            let $url$.fragment := $requestUrl$.fragment
        let $method'$ := $request$.method
        let $body'$ := $request$.body
        if Origin $\in$ $request$.headers then
            let $origin$ := $\langle$$request$.headers[Origin], $\langle$$request$.host, $url$.protocol$\rangle\rangle$
        else
            let $origin$ := $\perp$
        if $response$.status $\equiv 303 \wedge$ $request$.method $\notin$ {GET, HEAD} then
            let $method'$ := GET
            let $body'$ := $\langle\rangle$
```

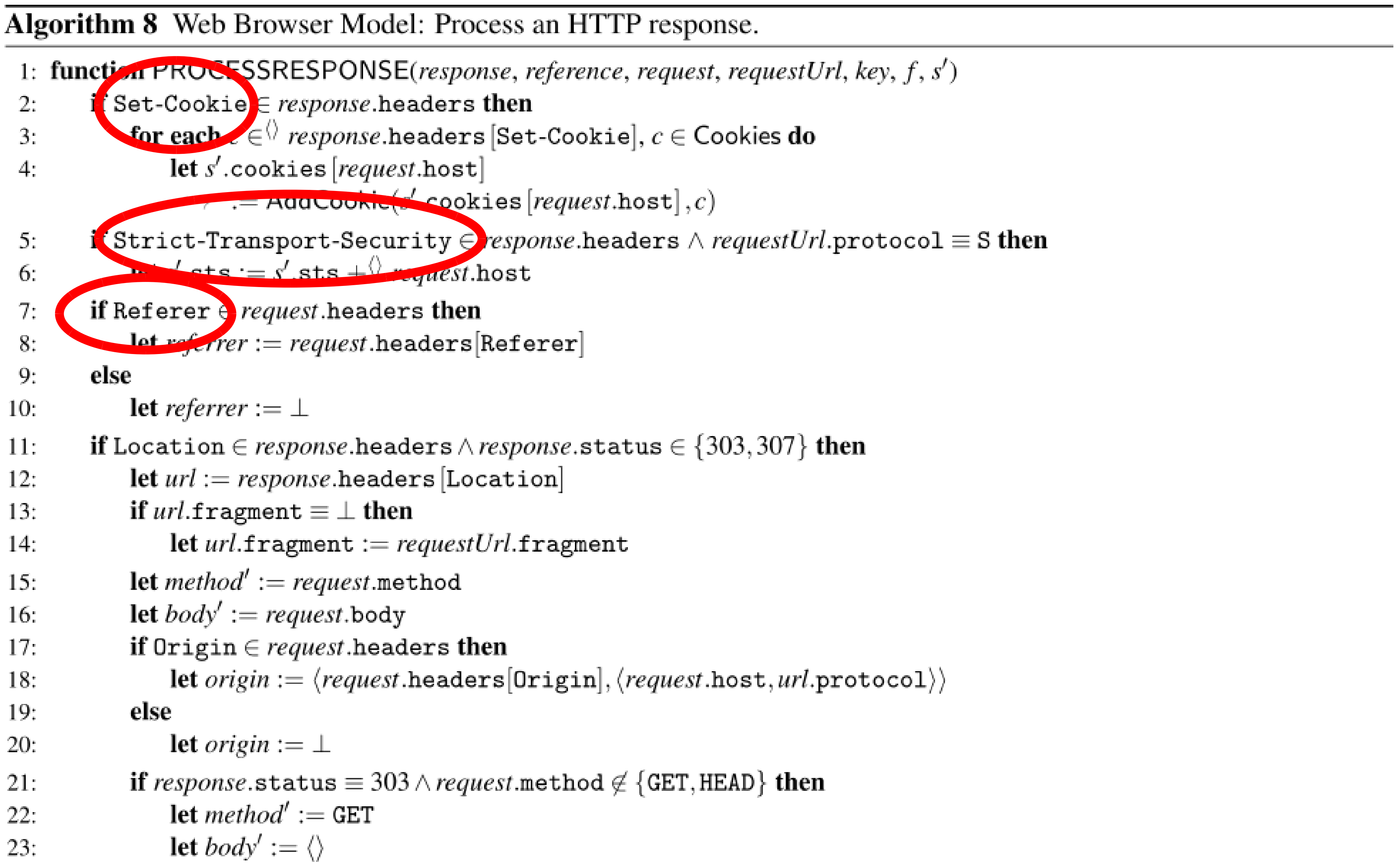
WIM: Web Browser Model – Excerpt



Algorithm 8 Web Browser Model: Process an HTTP response.

```
function PROCESSRESPONSE($response$, $reference$, $request$, $requestUrl$, $key$, $f$, $s'$)
    if Set-Cookie $\in$ $response$.headers then
        for each $c \in^{\langle\rangle}$ $response$.headers[Set-Cookie], $c \in$ Cookies do
            let $s'$.cookies[$request$.host]
                := AddCookie($s'$.cookies[$request$.host], $c$)
    if Strict-Transport-Security $\in$ $response$.headers $\wedge$ $requestUrl$.protocol $\equiv$ S then
        let $s'$.sts := $s'$.sts $+^{\langle\rangle}$ $request$.host
    if Referer $\in$ $request$.headers then
        let $referrer$ := $request$.headers[Referer]
    else
        let $referrer$ := $\perp$
    if Location $\in$ $response$.headers $\wedge$ $response$.status $\in \{303, 307\}$ then
        let $url$ := $response$.headers[Location]
        if $url$.fragment $\equiv \perp$ then
            let $url$.fragment := $requestUrl$.fragment
        let $method'$ := $request$.method
        let $body'$ := $request$.body
        if Origin $\in$ $request$.headers then
            let $origin$ := $\langle$$request$.headers[Origin], $\langle$$request$.host, $url$.protocol$\rangle\rangle$
        else
            let $origin$ := $\perp$
        if $response$.status $\equiv 303 \wedge$ $request$.method $\notin$ {GET, HEAD} then
            let $method'$ := GET
            let $body'$ := $\langle\rangle$
```

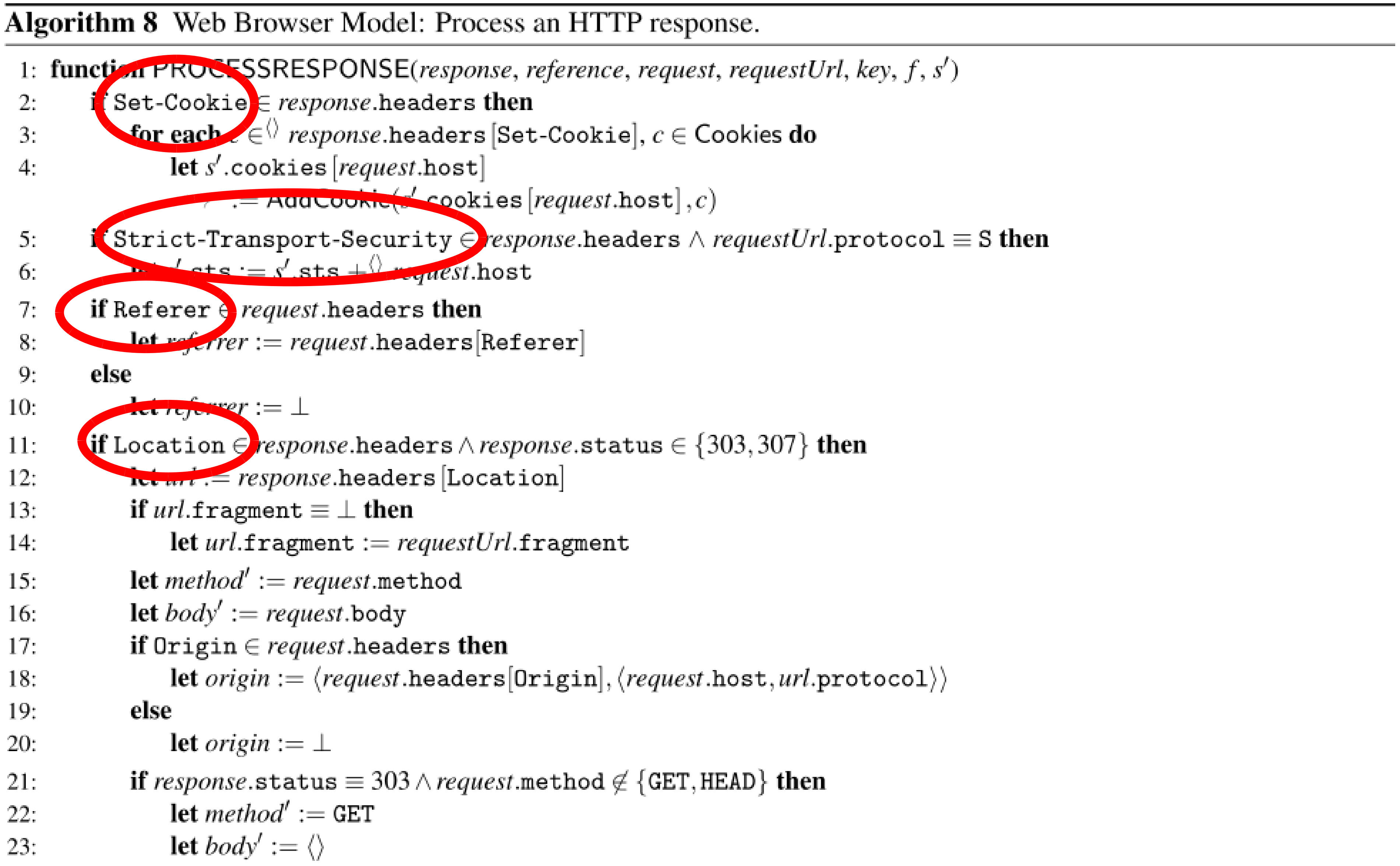
WIM: Web Browser Model – Excerpt



Algorithm 8 Web Browser Model: Process an HTTP response.

```
function PROCESSRESPONSE($response$, $reference$, $request$, $requestUrl$, $key$, $f$, $s'$)
    if Set-Cookie $\in$ $response$.headers then
        for each $c \in^{\langle\rangle}$ $response$.headers[Set-Cookie], $c \in$ Cookies do
            let $s'$.cookies[$request$.host]
                := AddCookie($s'$.cookies[$request$.host], $c$)
    if Strict-Transport-Security $\in$ $response$.headers $\wedge$ $requestUrl$.protocol $\equiv$ S then
        let $s'$.sts := $s'$.sts $+^{\langle\rangle}$ $request$.host
    if Referer $\in$ $request$.headers then
        let $referrer$ := $request$.headers[Referer]
    else
        let $referrer$ := $\perp$
    if Location $\in$ $response$.headers $\wedge$ $response$.status $\in \{303, 307\}$ then
        let $url$ := $response$.headers[Location]
        if $url$.fragment $\equiv \perp$ then
            let $url$.fragment := $requestUrl$.fragment
        let $method'$ := $request$.method
        let $body'$ := $request$.body
        if Origin $\in$ $request$.headers then
            let $origin$ := $\langle request$.headers[Origin], $\langle request$.host, $url$.protocol$\rangle\rangle$
        else
            let $origin$ := $\perp$
        if $response$.status $\equiv 303 \wedge request$.method $\notin \{$GET, HEAD$\}$ then
            let $method'$ := GET
            let $body'$ := $\langle\rangle$
```

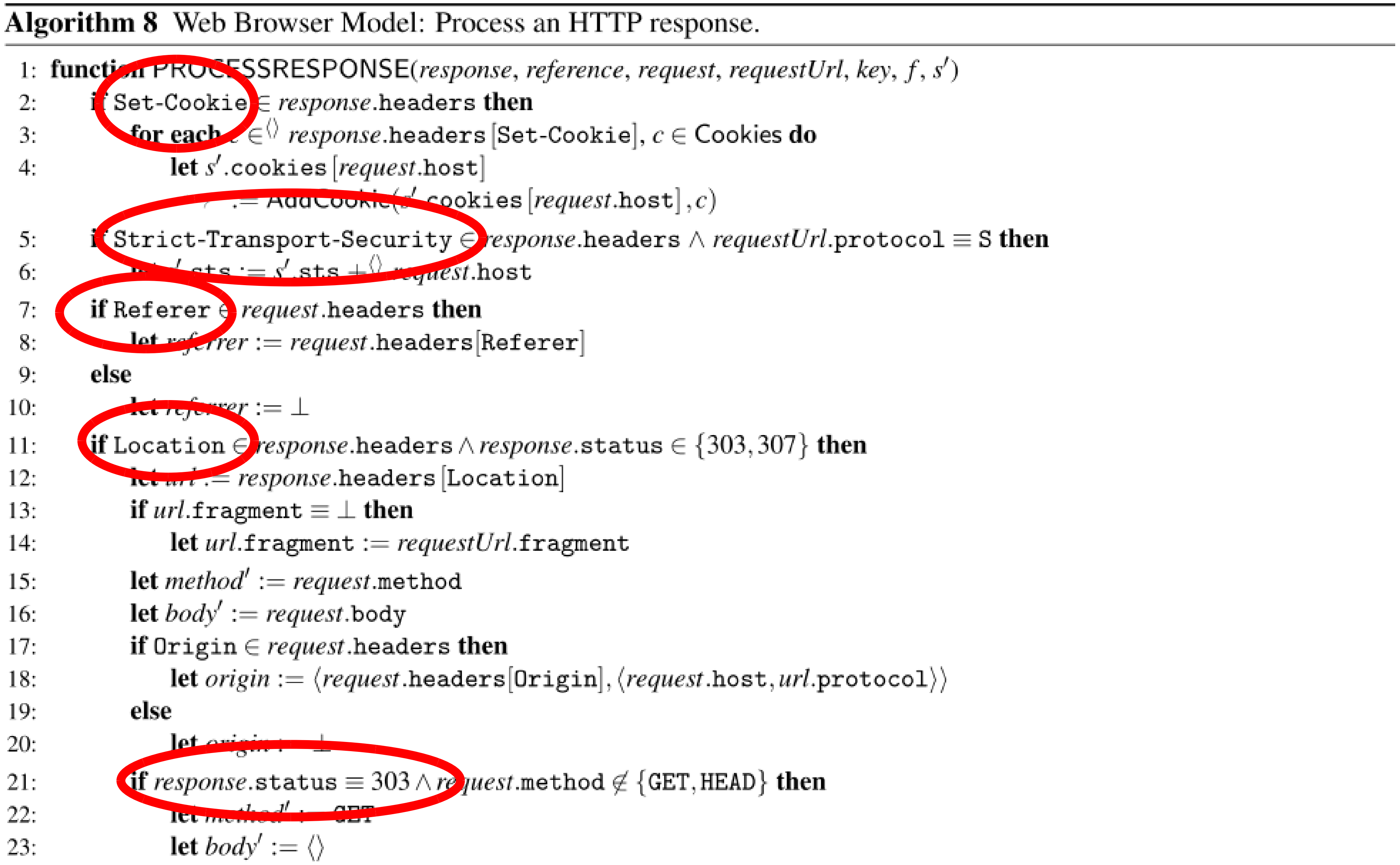
WIM: Web Browser Model – Excerpt



Algorithm 8 Web Browser Model: Process an HTTP response.

```
function PROCESSRESPONSE($response$, $reference$, $request$, $requestUrl$, $key$, $f$, $s'$)
    if Set-Cookie $\in$ $response$.headers then
        for each $c \in^{\langle\rangle}$ $response$.headers[Set-Cookie], $c \in$ Cookies do
            let $s'$.cookies[$request$.host]
                := AddCookie($s'$.cookies[$request$.host], $c$)
    if Strict-Transport-Security $\in$ $response$.headers $\wedge$ $requestUrl$.protocol $\equiv$ S then
        let $s'$.sts := $s'$.sts $+^{\langle\rangle}$ $request$.host
    if Referer $\in$ $request$.headers then
        let $referrer$ := $request$.headers[Referer]
    else
        let $referrer$ := $\bot$
    if Location $\in$ $response$.headers $\wedge$ $response$.status $\in \{303, 307\}$ then
        let $url$ := $response$.headers[Location]
        if $url$.fragment $\equiv \bot$ then
            let $url$.fragment := $requestUrl$.fragment
        let $method'$ := $request$.method
        let $body'$ := $request$.body
        if Origin $\in$ $request$.headers then
            let $origin$ := $\langle$$request$.headers[Origin], $\langle$$request$.host, $url$.protocol$\rangle\rangle$
        else
            let $origin$ := $\bot$
        if $response$.status $\equiv 303 \wedge$ $request$.method $\notin$ {GET, HEAD} then
            let $method'$ := GET
            let $body'$ := $\langle\rangle$
```

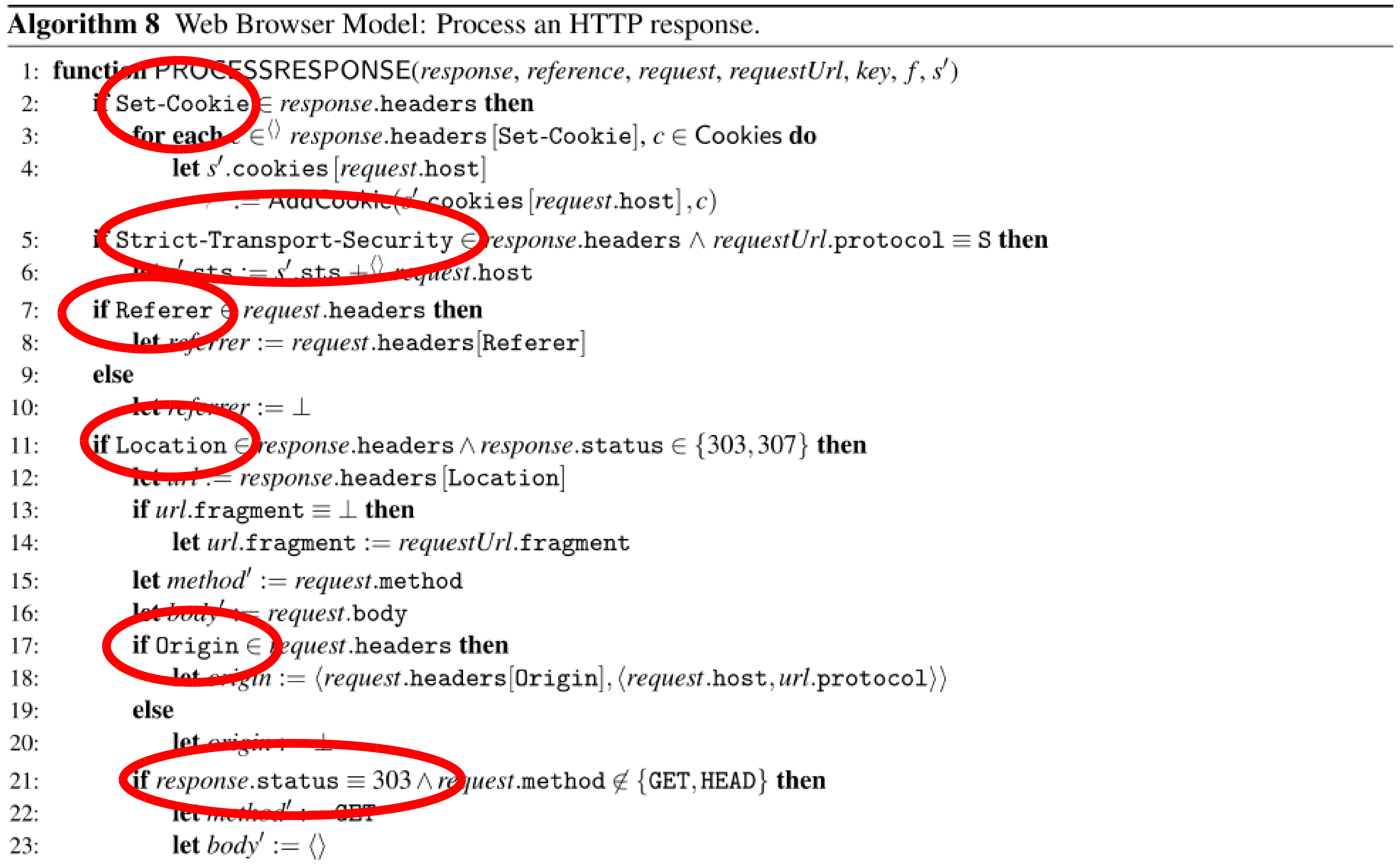
WIM: Web Browser Model – Excerpt



Algorithm 8 Web Browser Model: Process an HTTP response.

```
function ProcessResponse($response$, $reference$, $request$, $requestUrl$, $key$, $f$, $s'$)
    if $\mathtt{Set\text{-}Cookie} \in response.\mathtt{headers}$ then
        for each $c \in^{\langle\rangle} response.\mathtt{headers}[\mathtt{Set\text{-}Cookie}]$, $c \in \mathsf{Cookies}$ do
            let $s'.\mathtt{cookies}[request.\mathtt{host}]$
                $:= \mathsf{AddCookie}(s'.\mathtt{cookies}[request.\mathtt{host}], c)$
    if $\mathtt{Strict\text{-}Transport\text{-}Security} \in response.\mathtt{headers} \wedge requestUrl.\mathtt{protocol} \equiv \mathtt{S}$ then
        let $s'.\mathtt{sts} := s'.\mathtt{sts} +^{\langle\rangle} request.\mathtt{host}$
    if $\mathtt{Referer} \in request.\mathtt{headers}$ then
        let $referrer := request.\mathtt{headers}[\mathtt{Referer}]$
    else
        let $referrer := \perp$
    if $\mathtt{Location} \in response.\mathtt{headers} \wedge response.\mathtt{status} \in \{303, 307\}$ then
        let $url := response.\mathtt{headers}[\mathtt{Location}]$
        if $url.\mathtt{fragment} \equiv \perp$ then
            let $url.\mathtt{fragment} := requestUrl.\mathtt{fragment}$
        let $method' := request.\mathtt{method}$
        let $body' := request.\mathtt{body}$
        if $\mathtt{Origin} \in request.\mathtt{headers}$ then
            let $origin := \langle request.\mathtt{headers}[\mathtt{Origin}], \langle request.\mathtt{host}, url.\mathtt{protocol}\rangle\rangle$
        else
            let $origin := \perp$
        if $response.\mathtt{status} \equiv 303 \wedge request.\mathtt{method} \not\in \{\mathtt{GET}, \mathtt{HEAD}\}$ then
            let $method' := \mathtt{GET}$
            let $body' := \langle\rangle$
```

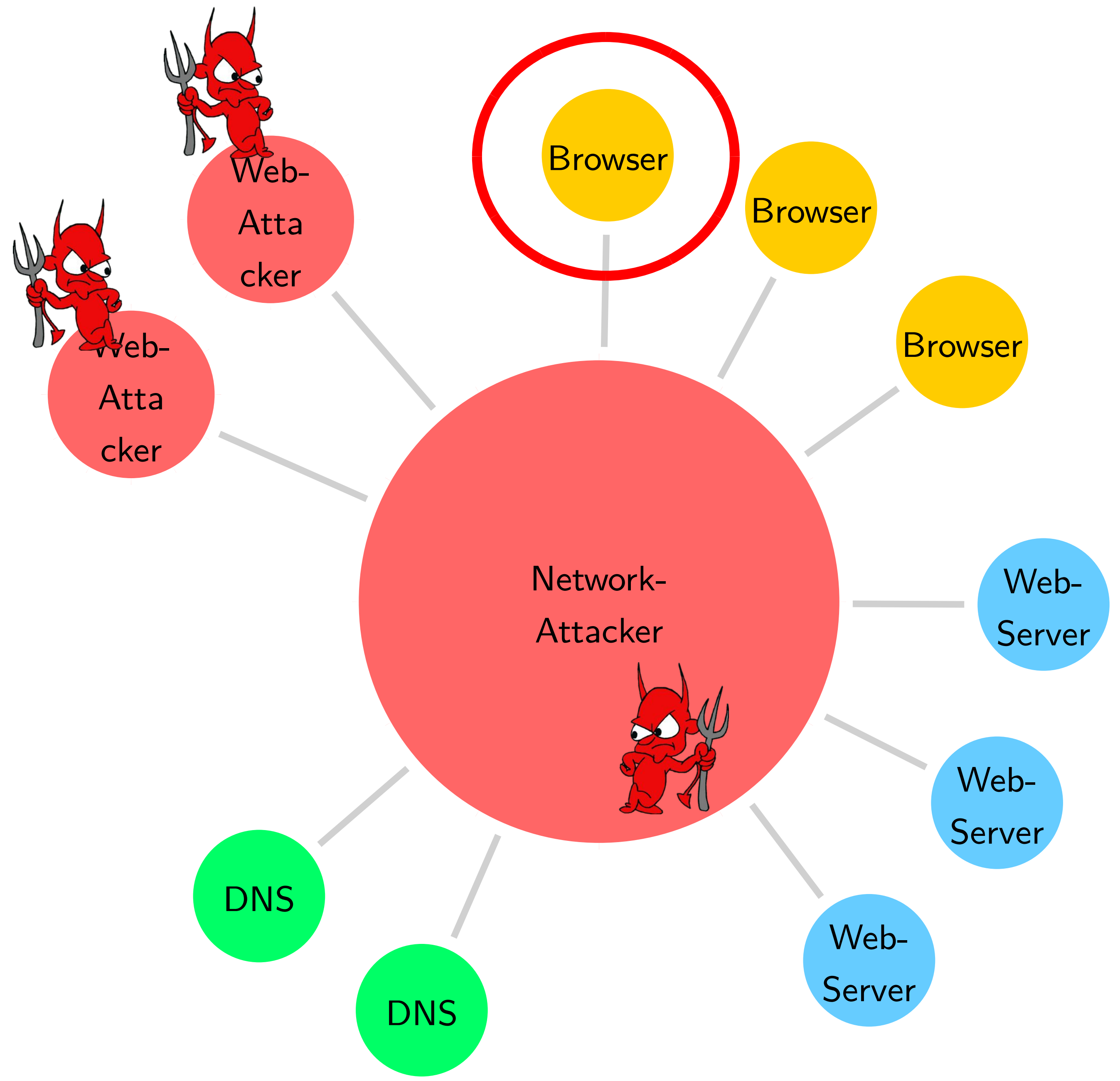
WIM: Web Browser Model – Excerpt



Algorithm 8 Web Browser Model: Process an HTTP response.

```
function PROCESSRESPONSE($response$, $reference$, $request$, $requestUrl$, $key$, $f$, $s'$)
    if Set-Cookie $\in$ $response$.headers then
        for each $c \in^{\langle\rangle}$ $response$.headers[Set-Cookie], $c \in$ Cookies do
            let $s'$.cookies[$request$.host]
                := AddCookie($s'$.cookies[$request$.host], $c$)
    if Strict-Transport-Security $\in$ $response$.headers $\wedge$ $requestUrl$.protocol $\equiv$ S then
        let $s'$.sts := $s'$.sts $+^{\langle\rangle}$ $request$.host
    if Referer $\in$ $request$.headers then
        let $referrer$ := $request$.headers[Referer]
    else
        let $referrer$ := $\perp$
    if Location $\in$ $response$.headers $\wedge$ $response$.status $\in \{303, 307\}$ then
        let $url$ := $response$.headers[Location]
        if $url$.fragment $\equiv \perp$ then
            let $url$.fragment := $requestUrl$.fragment
        let $method'$ := $request$.method
        let $body'$ := $request$.body
        if Origin $\in$ $request$.headers then
            let $origin$ := $\langle$$request$.headers[Origin], $\langle$$request$.host, $url$.protocol$\rangle\rangle$
        else
            let $origin$ := $\perp$
        if $response$.status $\equiv 303 \wedge$ $request$.method $\notin$ {GET, HEAD} then
            let $method'$ := GET
            let $body'$ := $\langle\rangle$
```

WIM: Network Model and Attackers



Dolev-Yao-Attacker

Limitations

- No language details
- No user interface details (e.g., no clickjacking attacks)
- No byte-level attacks (e.g., buffer overflows)
- Abstract view on cryptography and TLS

Model can in principle be extended to capture these aspects as well.

Trade-off: comprehensiveness vs. simplicity

Limitations

- No language details

main focus in other work,
e.g., Calzavara, Foccardi et al.

- No user interface details (e.g., no clickjacking attacks)
- No byte-level attacks (e.g., buffer overflows)
- Abstract view on cryptography and TLS

Model can in principle be extended to capture these aspects as well.
Trade-off: comprehensiveness vs. simplicity

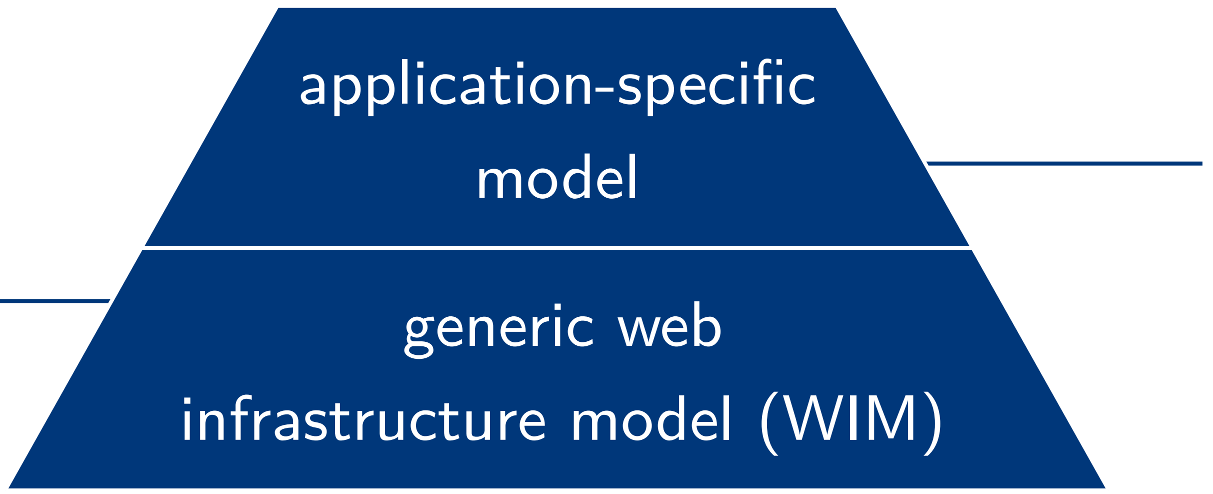
How to use the WIM?

How to use the WIM?

Foundation:
Formal description
of the web

generic web
infrastructure model (WIM)

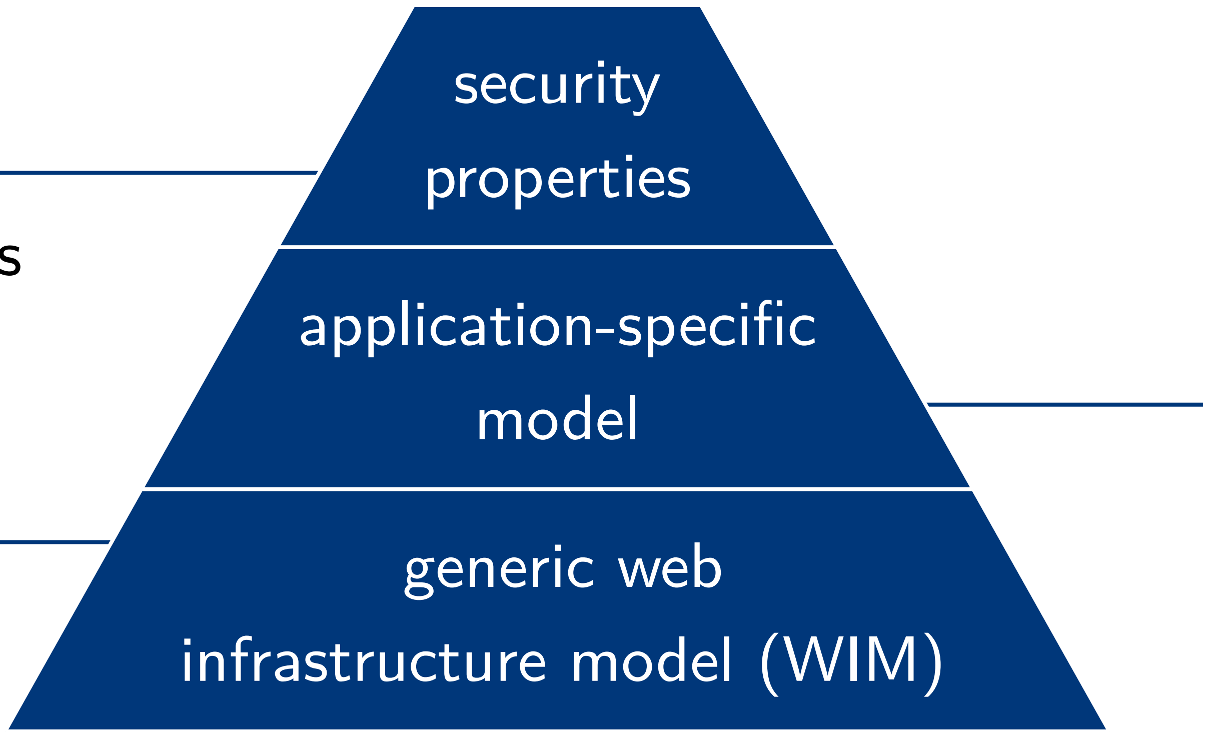
How to use the WIM?


application-specific model
generic web infrastructure model (WIM)

Foundation:
Formal description of the web

Application model built from source code or specification

How to use the WIM?

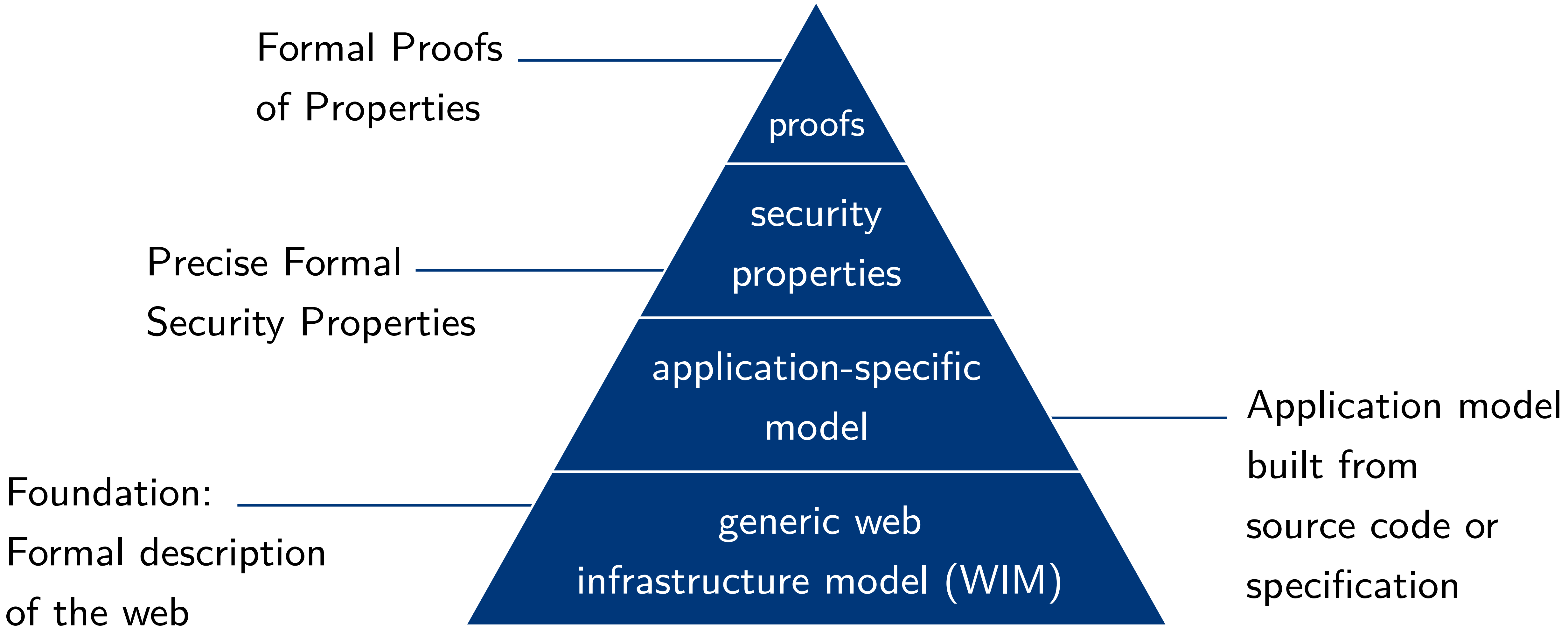


Precise Formal
Security Properties

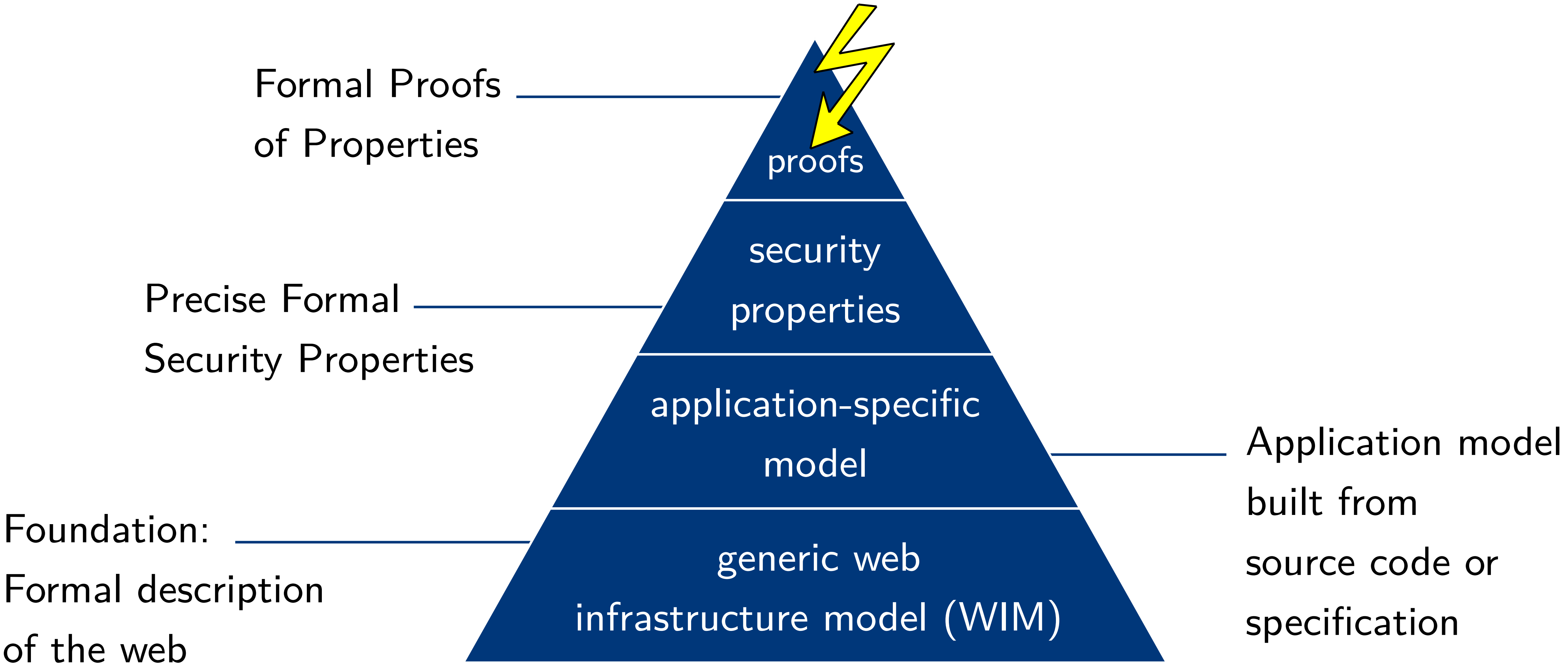
Foundation:
Formal description
of the web

Application model
built from
source code or
specification

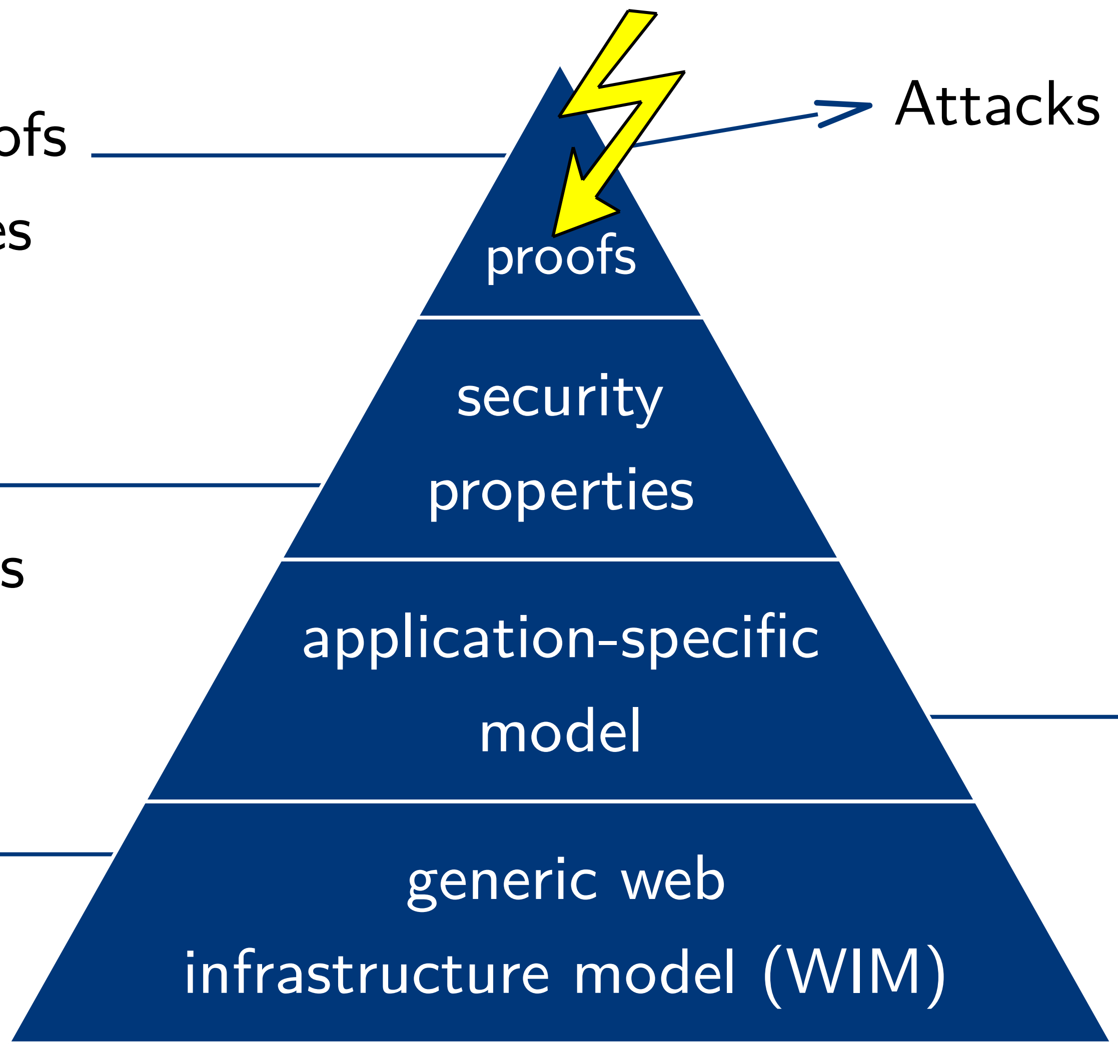
How to use the WIM?


Formal Proofs of Properties
proofs
Precise Formal Security Properties
security properties
application-specific model
Application model built from source code or specification
Foundation: Formal description of the web
generic web infrastructure model (WIM)

How to use the WIM?



How to use the WIM?


proofs
security properties
application-specific model
generic web infrastructure model (WIM)

Formal Proofs of Properties

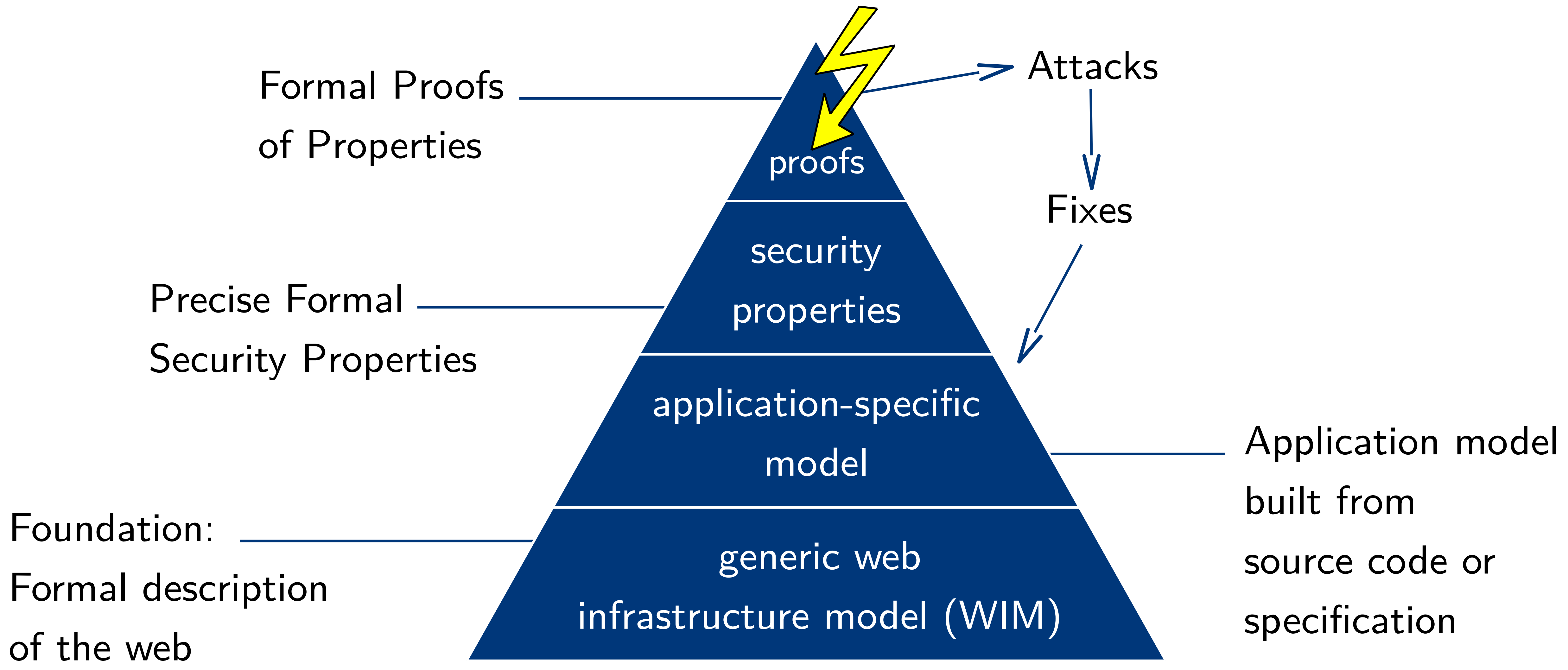
Attacks

Precise Formal Security Properties

Application model built from source code or specification

Foundation: Formal description of the web

How to use the WIM?

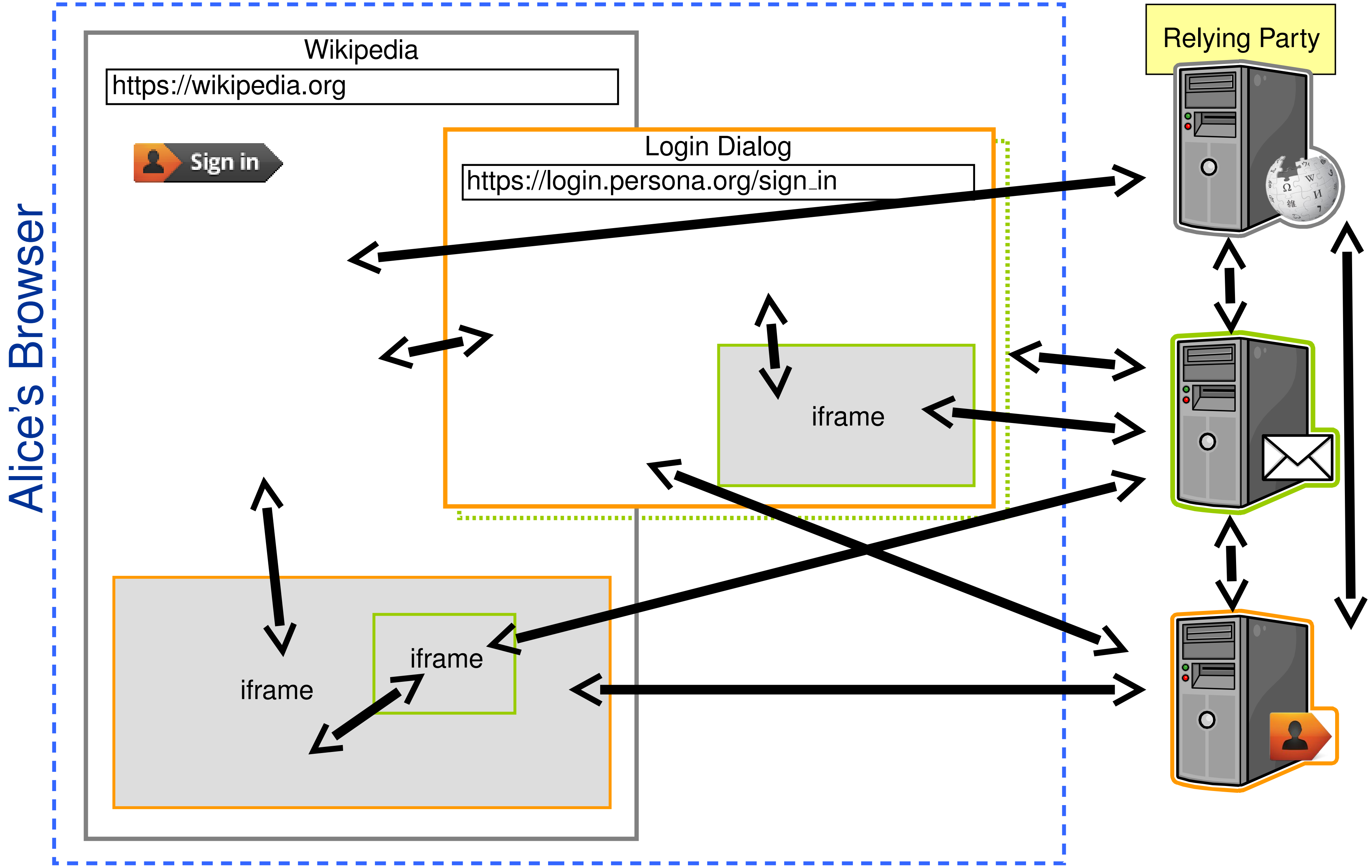

Formal Proofs of Properties
Precise Formal Security Properties
Foundation: Formal description of the web
proofs
security properties
application-specific model
generic web infrastructure model (WIM)
Attacks
Fixes
Application model built from source code or specification

Case Studies

Case Studies – Our Very First WIM Case Study [S&P14]

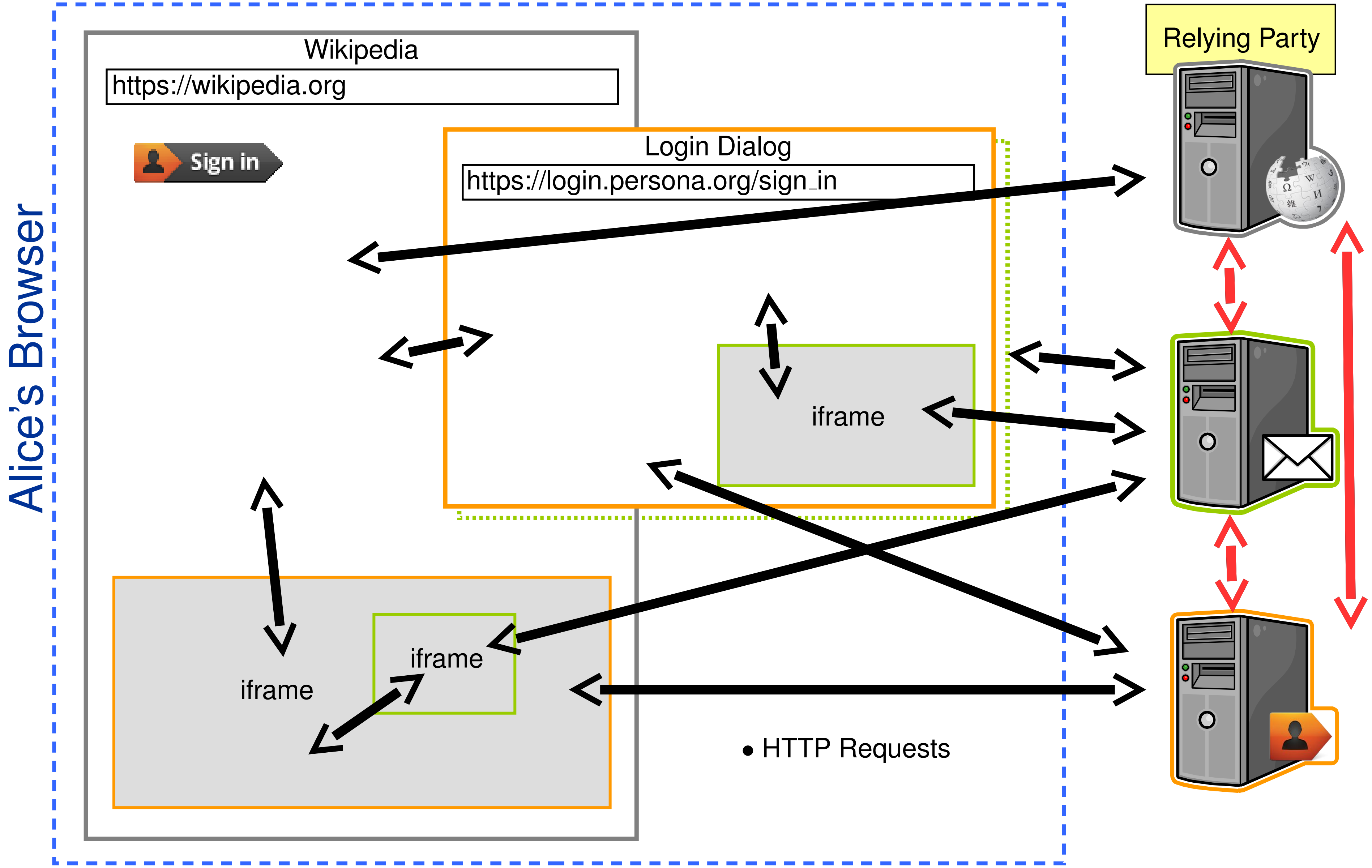
Example: Mozilla BrowserID

- BrowserID used a complex system of iframes and windows to transfer data between the RP, the IdP, and Mozilla's own servers.
- This was supposed to hide RP's identity from the IdP (but not from Mozilla).



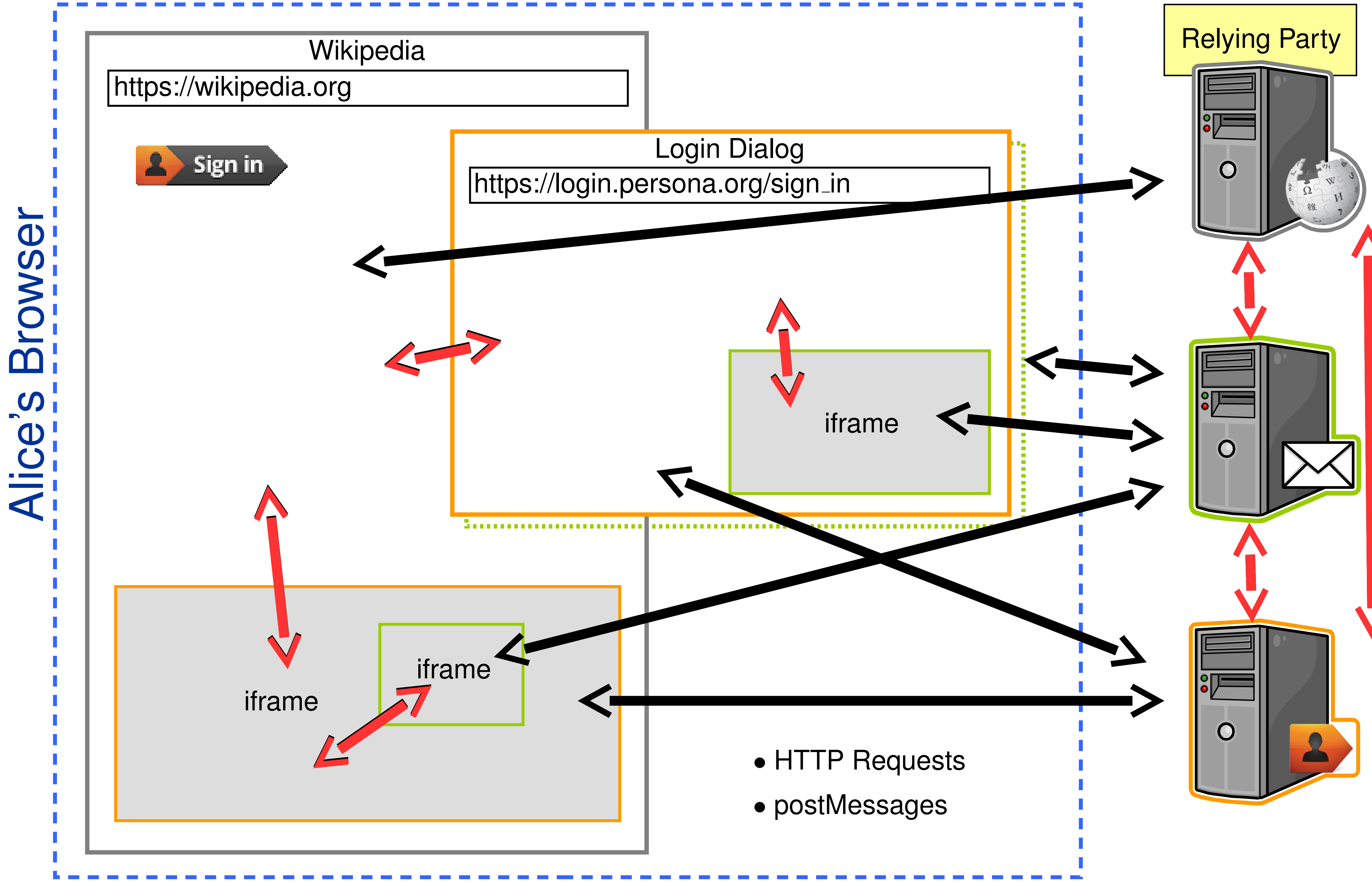
Example: Mozilla BrowserID

- BrowserID used a complex system of iframes and windows to transfer data between the RP, the IdP, and Mozilla's own servers.
- This was supposed to hide RP's identity from the IdP (but not from Mozilla).



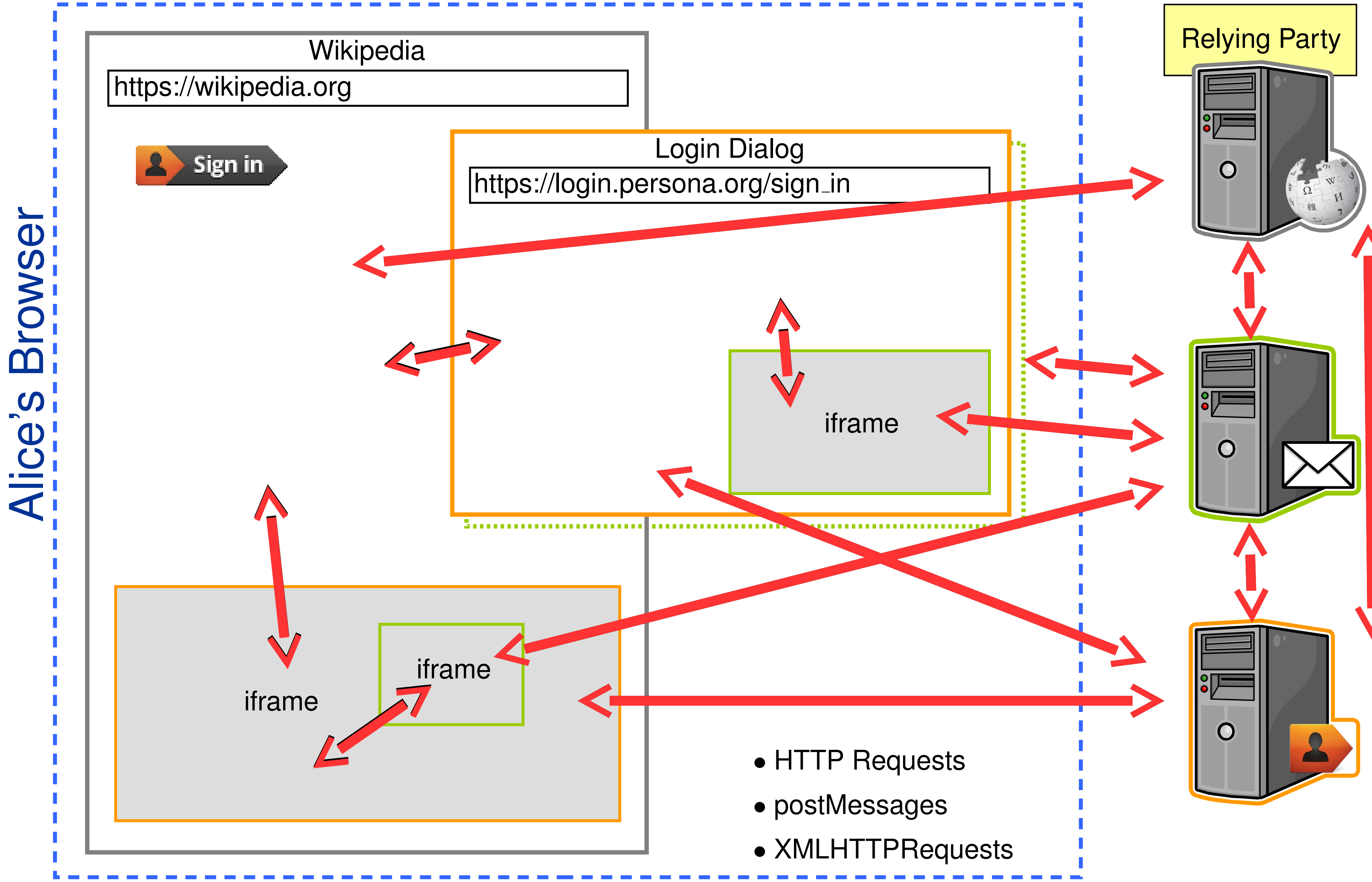
Example: Mozilla BrowserID

- BrowserID used a complex system of iframes and windows to transfer data between the RP, the IdP, and Mozilla's own servers.
- This was supposed to hide RP's identity from the IdP (but not from Mozilla).



Example: Mozilla BrowserID

- BrowserID used a complex system of iframes and windows to transfer data between the RP, the IdP, and Mozilla's own servers.
- This was supposed to hide RP's identity from the IdP (but not from Mozilla).



Results

- Analysis of Mozilla's BrowserID (a.k.a. Mozilla Persona) [SP2014, ESORICS2015]
  Main design goal: privacy
  - Found severe attacks: Identity Injection Attack, PostMessage-Based Attack,
  - Proposed fixes for authentication and proved security
  - Privacy broken beyond repair
- Designed our own new SSO system: SPRESSO (https://spresso.me) [CCS2015]
  First provably secure SSO system that provides strong authentication and privacy properties.

Case Studies – The Obvious Next Targets: OAuth and OpenID Connect [CCS 2016; CSF 2017]

OAuth 2.0 and OpenID Connect WIM Analyses

"Probably not interesting, too many other people have looked at this."

PhD Students

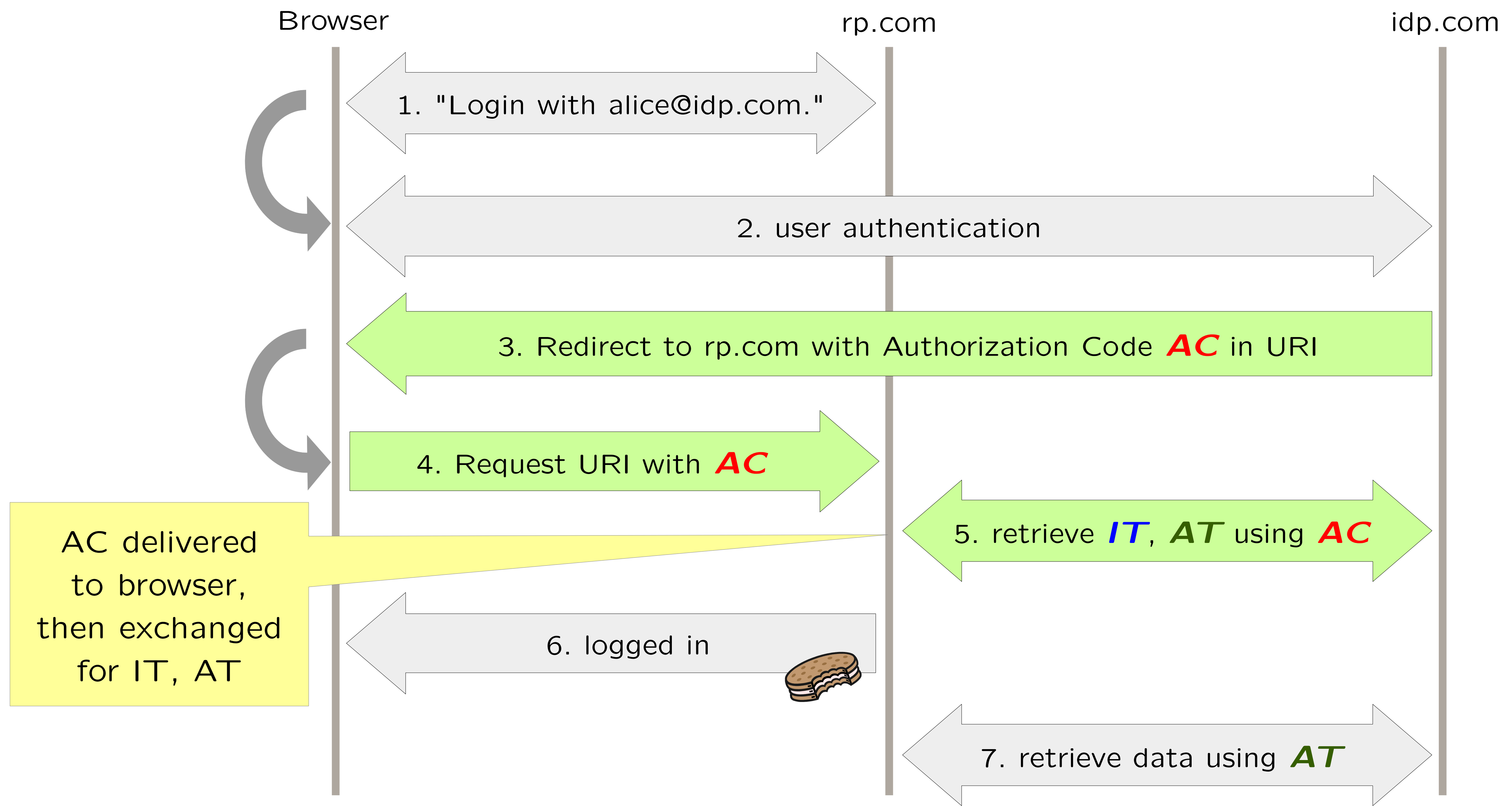
OAuth 2.0 and OpenID Connect WIM Analyses

“Probably not interesting, too many other people have looked at this.”

PhD Students

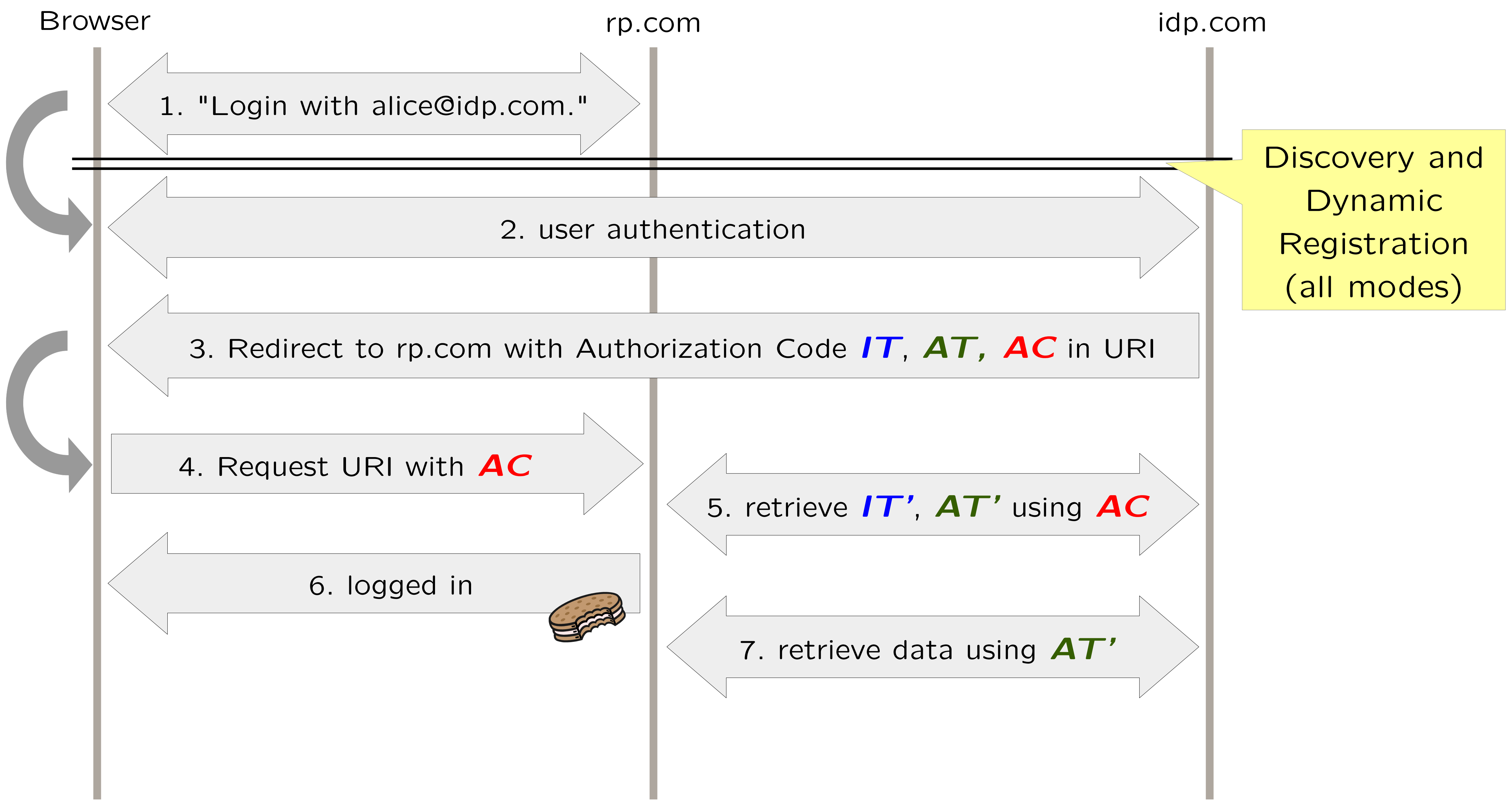
... I insisted

Authorization Code Mode

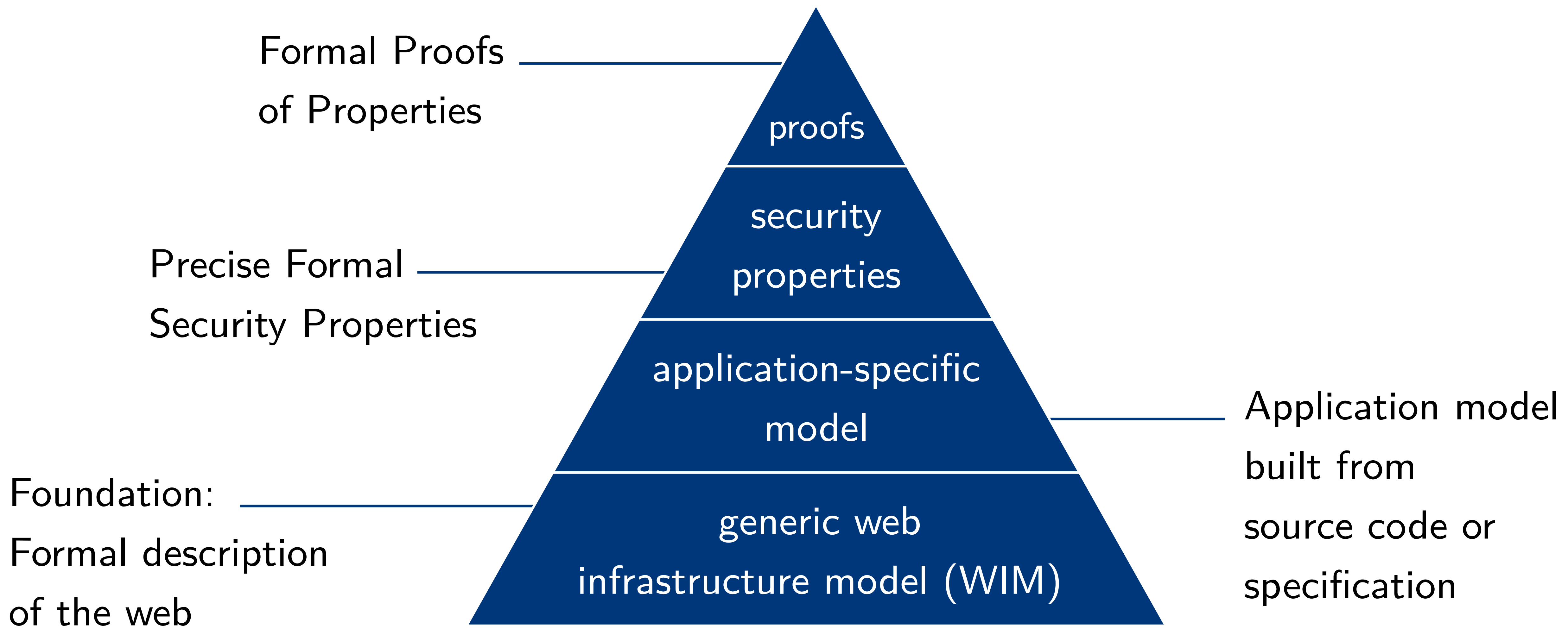


Browser
rp.com
idp.com
1. "Login with alice@idp.com."
2. user authentication
3. Redirect to rp.com with Authorization Code AC in URI
4. Request URI with AC
5. retrieve IT, AT using AC
6. logged in
7. retrieve data using AT
AC delivered to browser, then exchanged for IT, AT

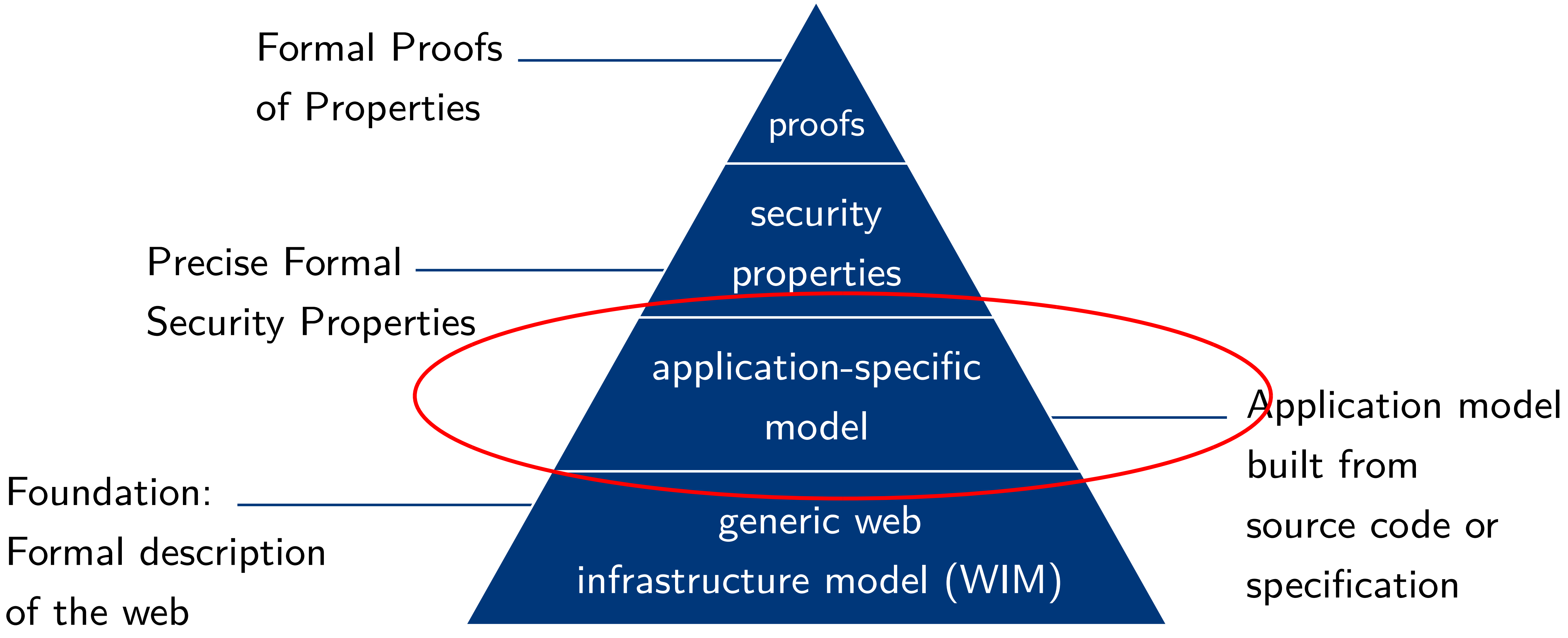
Discovery and Dynamic Registration


Browser
rp.com
idp.com
1. "Login with alice@idp.com."
Discovery and Dynamic Registration (all modes)
2. user authentication
3. Redirect to rp.com with Authorization Code IT, AT, AC in URI
4. Request URI with AC
5. retrieve IT', AT' using AC
6. logged in
7. retrieve data using AT'

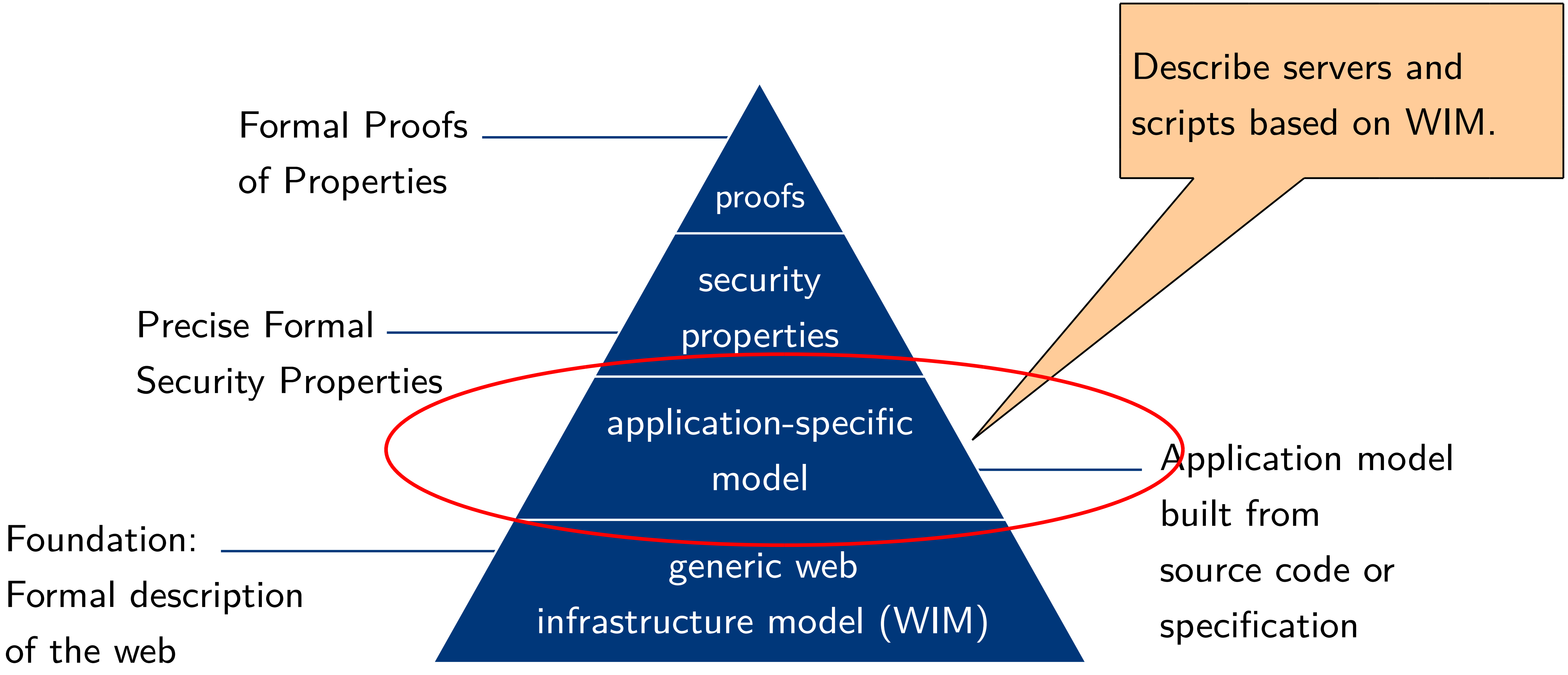
How to use the WIM?


Formal Proofs of Properties
Precise Formal Security Properties
Foundation: Formal description of the web
proofs
security properties
application-specific model
generic web infrastructure model (WIM)
Application model built from source code or specification

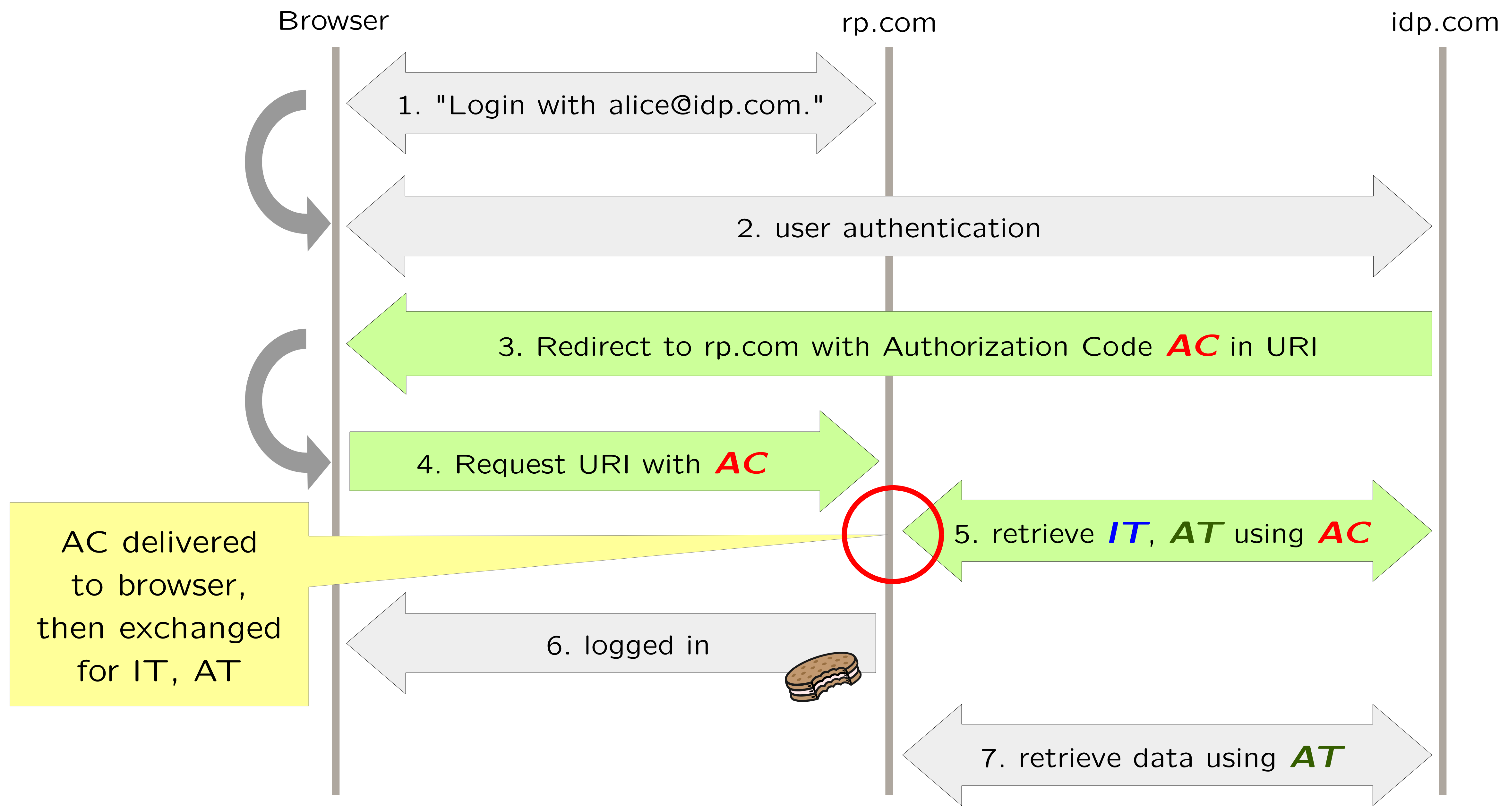
How to use the WIM?


Formal Proofs of Properties
proofs
Precise Formal Security Properties
security properties
application-specific model
Application model built from source code or specification
Foundation: Formal description of the web
generic web infrastructure model (WIM)

How to use the WIM?


Formal Proofs of Properties
Precise Formal Security Properties
Foundation: Formal description of the web
proofs
security properties
application-specific model
generic web infrastructure model (WIM)
Application model built from source code or specification
Describe servers and scripts based on WIM.

Authorization Code Mode



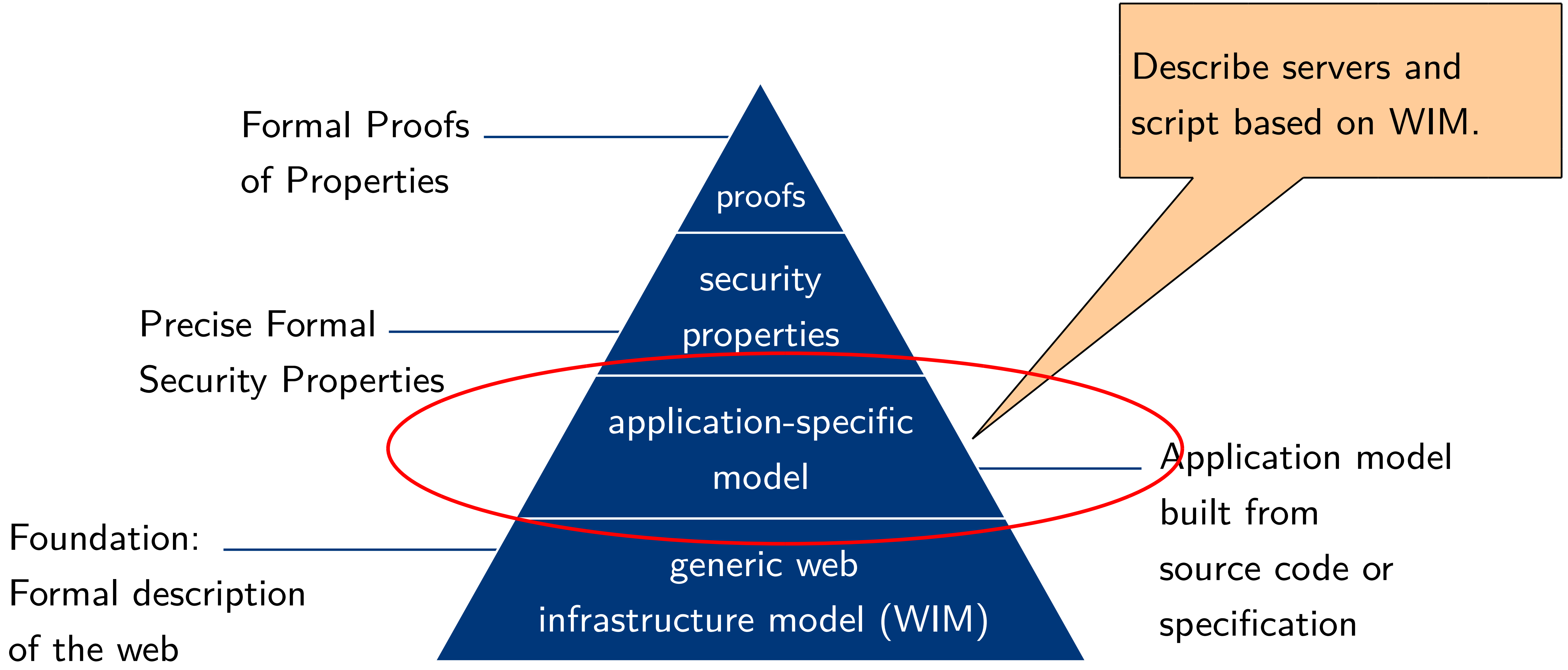
Browser
rp.com
idp.com
1. "Login with alice@idp.com."
2. user authentication
3. Redirect to rp.com with Authorization Code *AC* in URI
4. Request URI with *AC*
5. retrieve *IT*, *AT* using *AC*
6. logged in
7. retrieve data using *AT*
AC delivered to browser, then exchanged for IT, AT

Example: RP Checks an ID Token

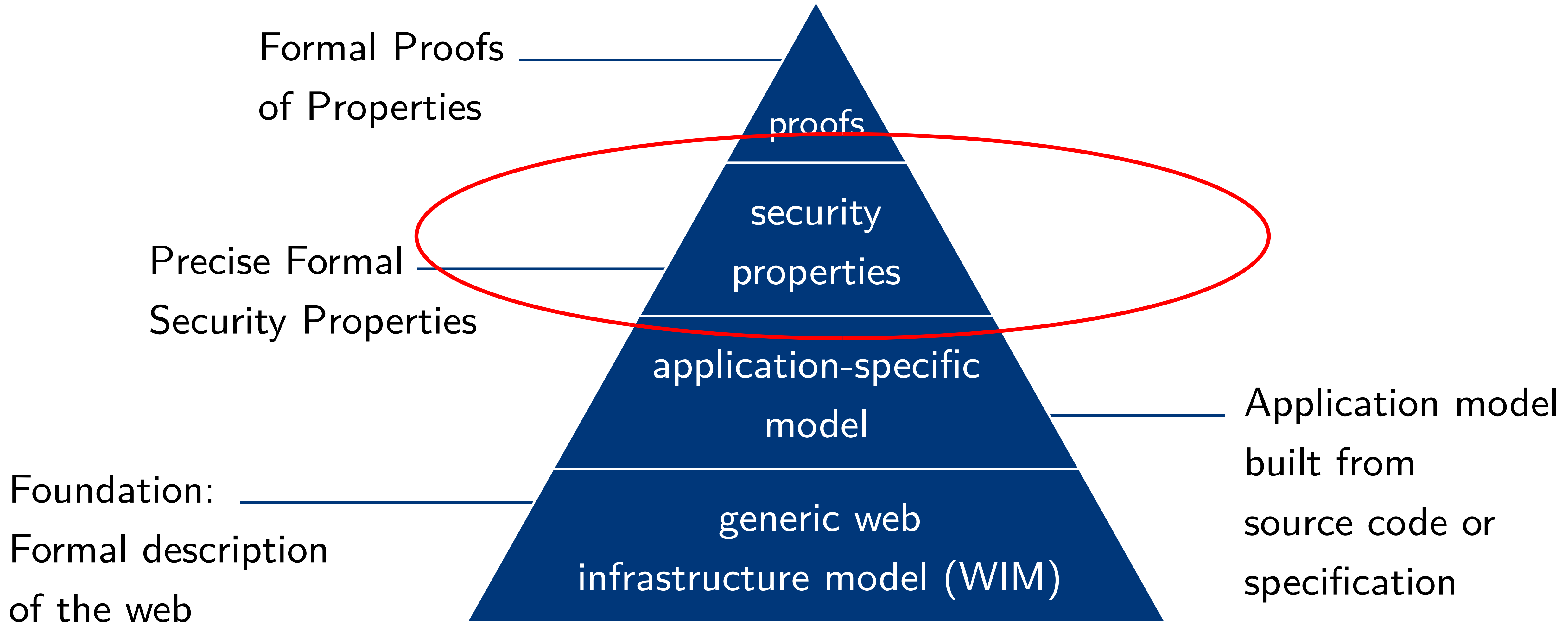
Algorithm 20 Relying Party R^r: Check id token.

```
1: function CHECK_ID_TOKEN($sessionId$, $id\_token$, $s'$)    → Check id token validity and create service session.
2:     let $session := s'$.sessions[$sessionId$]    → Retrieve session data.
3:     let $identity := session$[identity]
4:     let $issuer := s'$.issuerCache[$identity$]    → Retrieve issuer.
5:     let $oidcConfig := s'$.oidcConfigCache[$issuer$]    → Retrieve OIDC configuration for that issuer.
6:     let $credentials := s'$.clientCredentialsCache[$issuer$]    → Retrieve OIDC credentials for issuer.
7:     let $jwks := s'$.jwksCache[$issuer$]    → Retrieve signing keys for issuer.
8:     let $data :=$ extractmsg($id\_token$)    → Extract contents of signed id token.
9:     if $data$[iss] $\not\equiv issuer$ then
10:        stop    → Check the issuer.
11:    if $data$[aud] $\not\equiv credentials$[client_id] then
12:        stop    → Check the audience against own client id.
13:    if checksig($id\_token$,$jwks$) $\not\equiv \top$ then
14:        stop    → Check the signature of the id token.
15:    if $nonce \in session \wedge data$[nonce] $\not\equiv session$[nonce] then
16:        stop    → If a nonce was used, check its value.
17:    let $s'$.sessions[$sessionId$][loggedInAs] $:= \langle issuer, data$[sub]$\rangle$    → User is now logged in. Store user identity and issuer.
18:    let $s'$.sessions[$sessionId$][serviceSessionId] $:= \nu_4$    → Choose a new service session id.
19:    let $request := session$[redirectEpRequest]    → Retrieve stored meta data of the request from the browser to the redir. end-
                                                   point in order to respond to it now. The request's meta data was stored in
                                                   PROCESS_HTTPS_REQUEST (Algorithm 17).
20:    let $headers :=$ [ReferrerPolicy:origin]
21:    let $headers$[Set-Cookie] := [serviceSessionId:$\langle \nu_4, \top, \top, \top \rangle$]    → Create a cookie containing the service session id.
22:    let $m' := \mathsf{enc_s}(\langle$HTTPResp, $request$[message].nonce, 200, $headers$, ok$\rangle$, $request$[key])    → Respond to browser's request to the redirec-
                                                   tion endpoint.
23:    stop $\langle\langle request$[sender], $request$[receiver], $m'\rangle\rangle$, $s'$
```

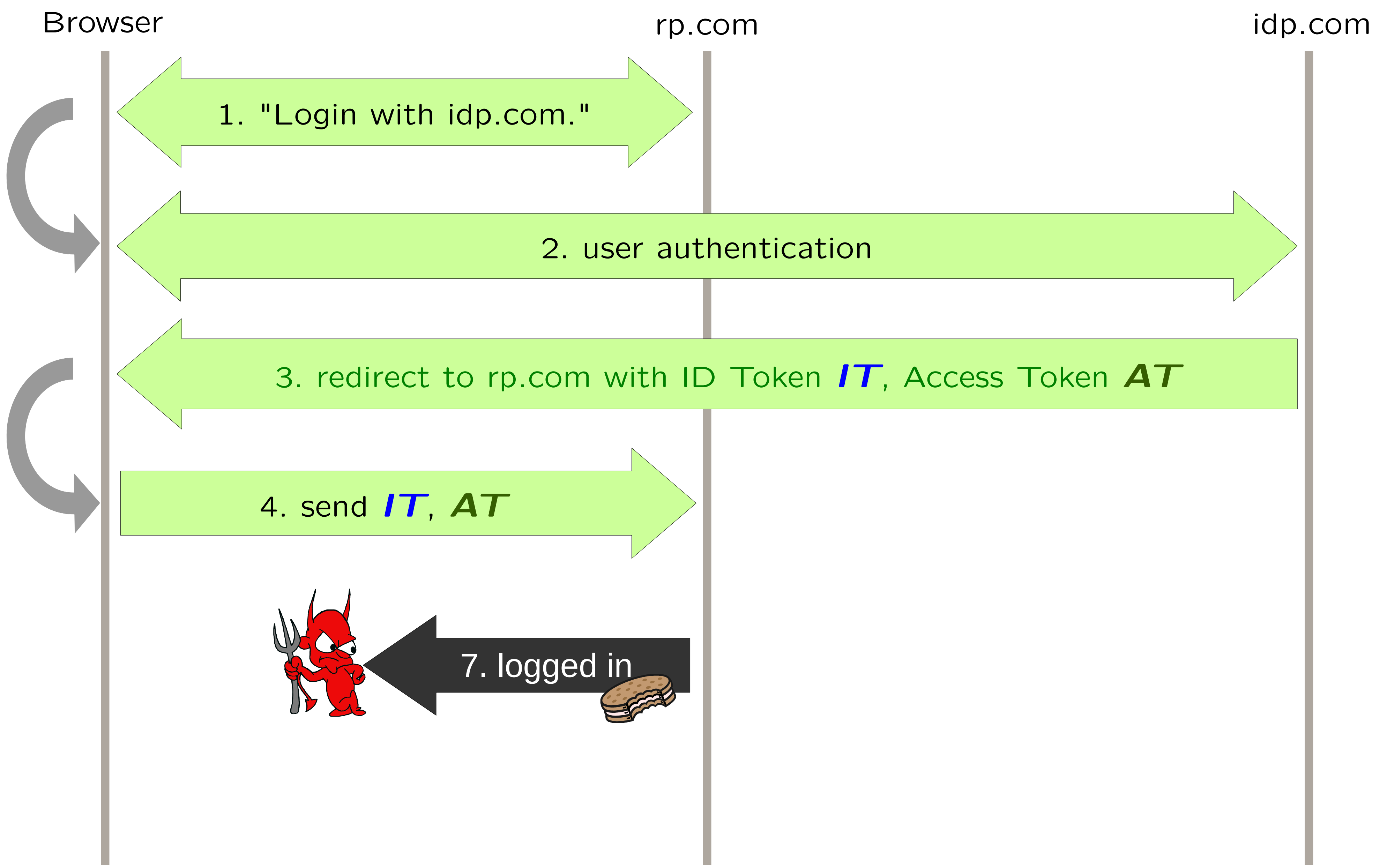
How to use the WIM?


Describe servers and script based on WIM.
Formal Proofs of Properties
proofs
security properties
Precise Formal Security Properties
application-specific model
Application model built from source code or specification
Foundation: Formal description of the web
generic web infrastructure model (WIM)

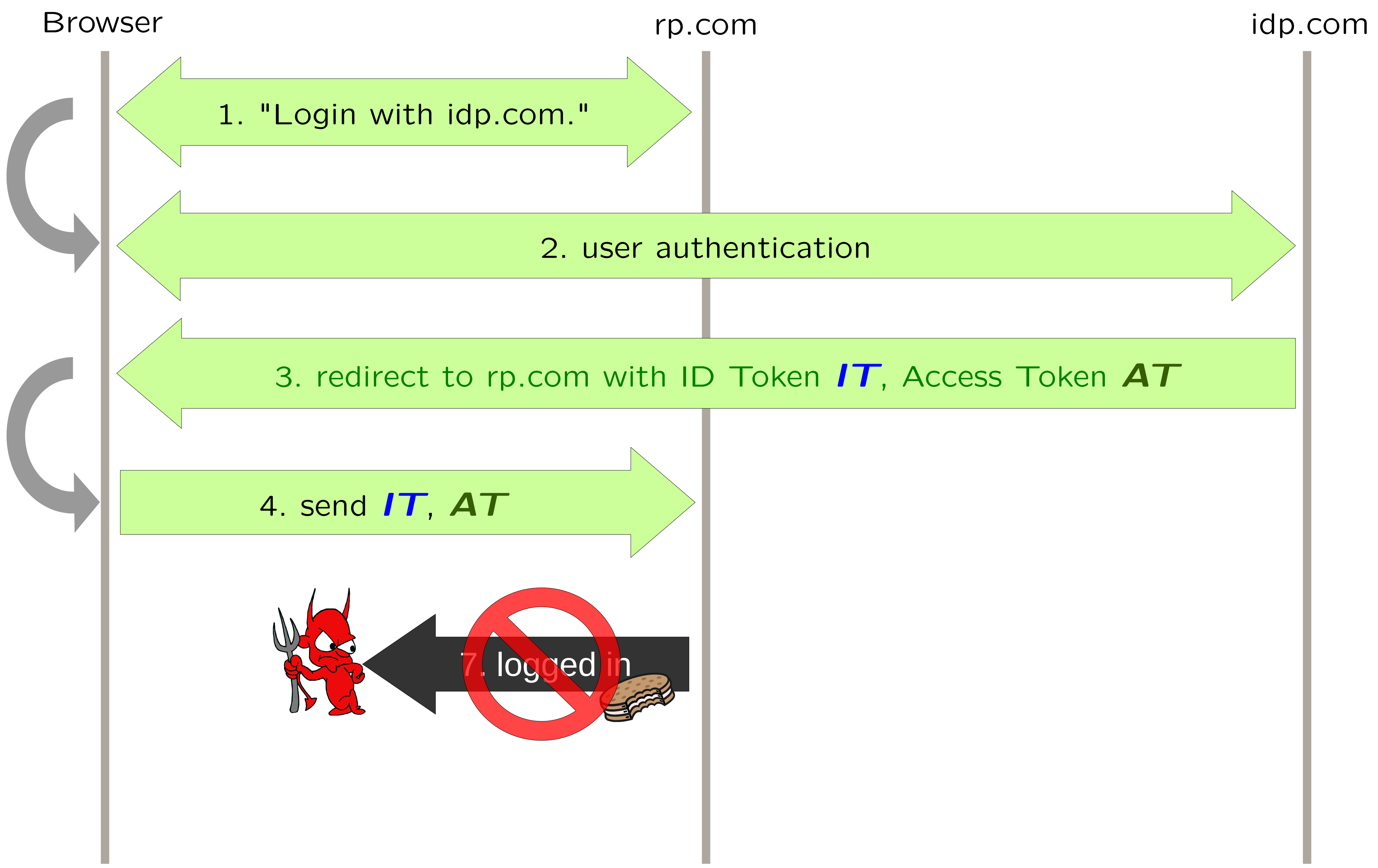
How to use the WIM?


Formal Proofs of Properties
proofs
security properties
Precise Formal Security Properties
application-specific model
Application model built from source code or specification
Foundation: Formal description of the web
generic web infrastructure model (WIM)

Authentication Property



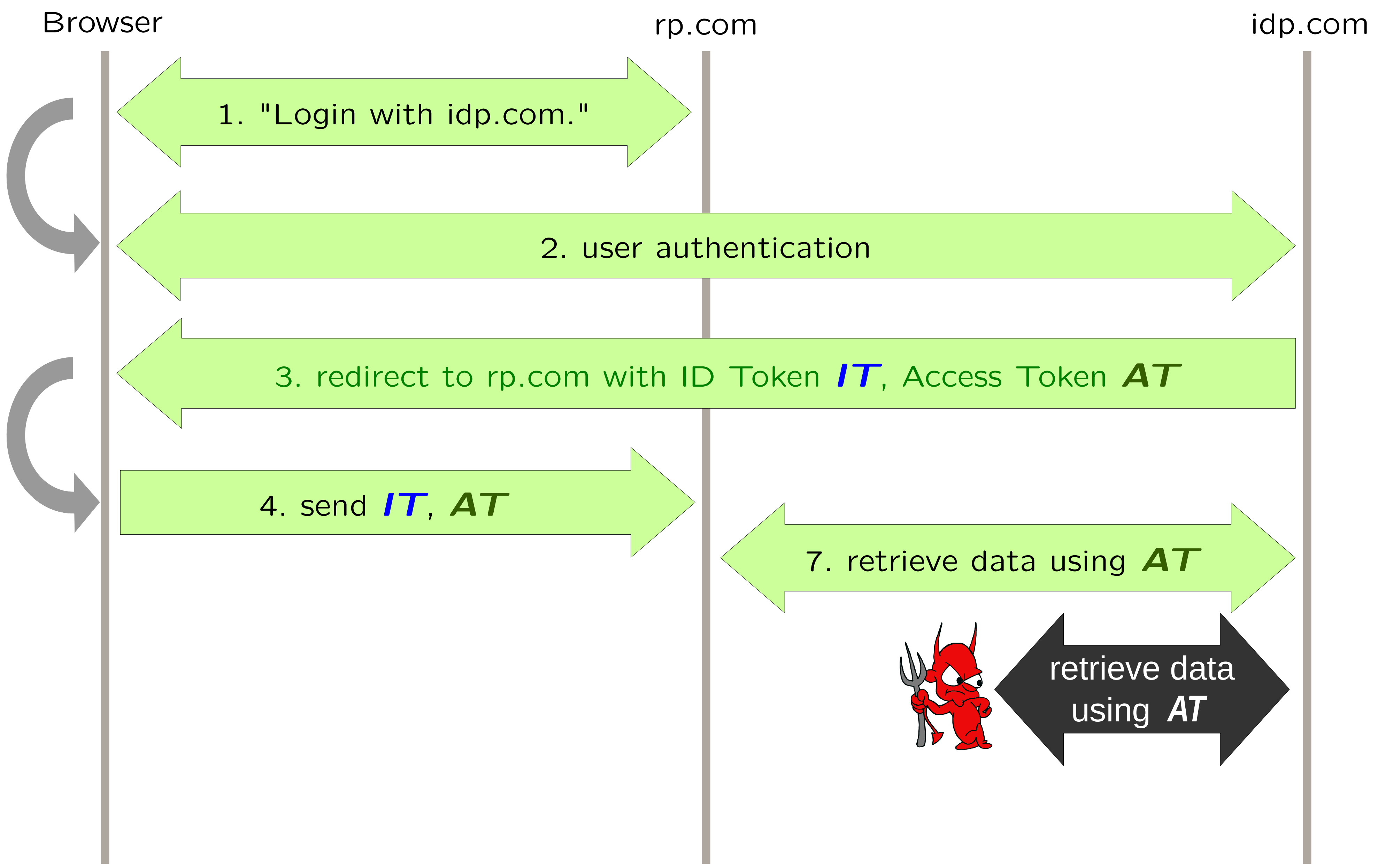
Authentication Property



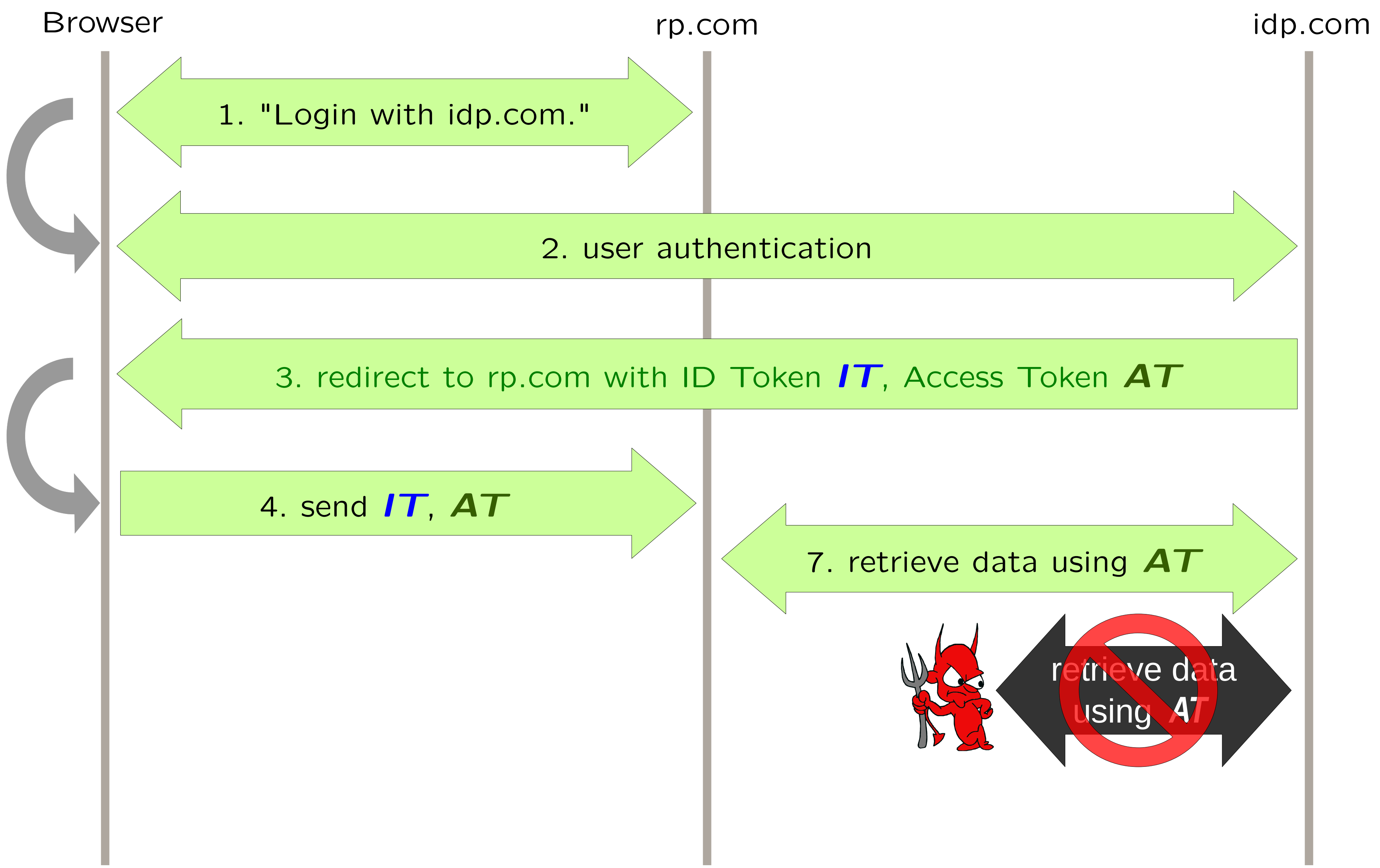
Authentication Property

Definition 46 (Authentication Property). Let $\mathcal{OWS}^n$ be an OAuth web system with a network attacker. We say that $\mathcal{OWS}^n$ *is secure w.r.t. authentication* iff for every run ρ of $\mathcal{OWS}^n$, every state (S^j, E^j, N^j) in ρ, every $r \in \mathsf{RP}$ that is honest in S^j, every $i \in \mathsf{IDP}$, every $g \in \mathsf{dom}(i)$, every $u \in \mathbb{S}$, every RP service token of the form $\langle n, \langle u, g\rangle\rangle$ recorded in $S^j(r).\mathtt{serviceTokens}$, and n being derivable from the attackers knowledge in S^j (i.e., $n \in d_{\emptyset}(S^j(\mathsf{attacker}))$), then the browser b owning u is fully corrupted in S^j (i.e., the value of *isCorrupted* is $\mathtt{FULLCORRUPT}$), some $r' \in \mathsf{trustedRPs}(\mathsf{secretOfID}(\langle u, g\rangle))$ is corrupted in S^j, or i is corrupted in S^j.

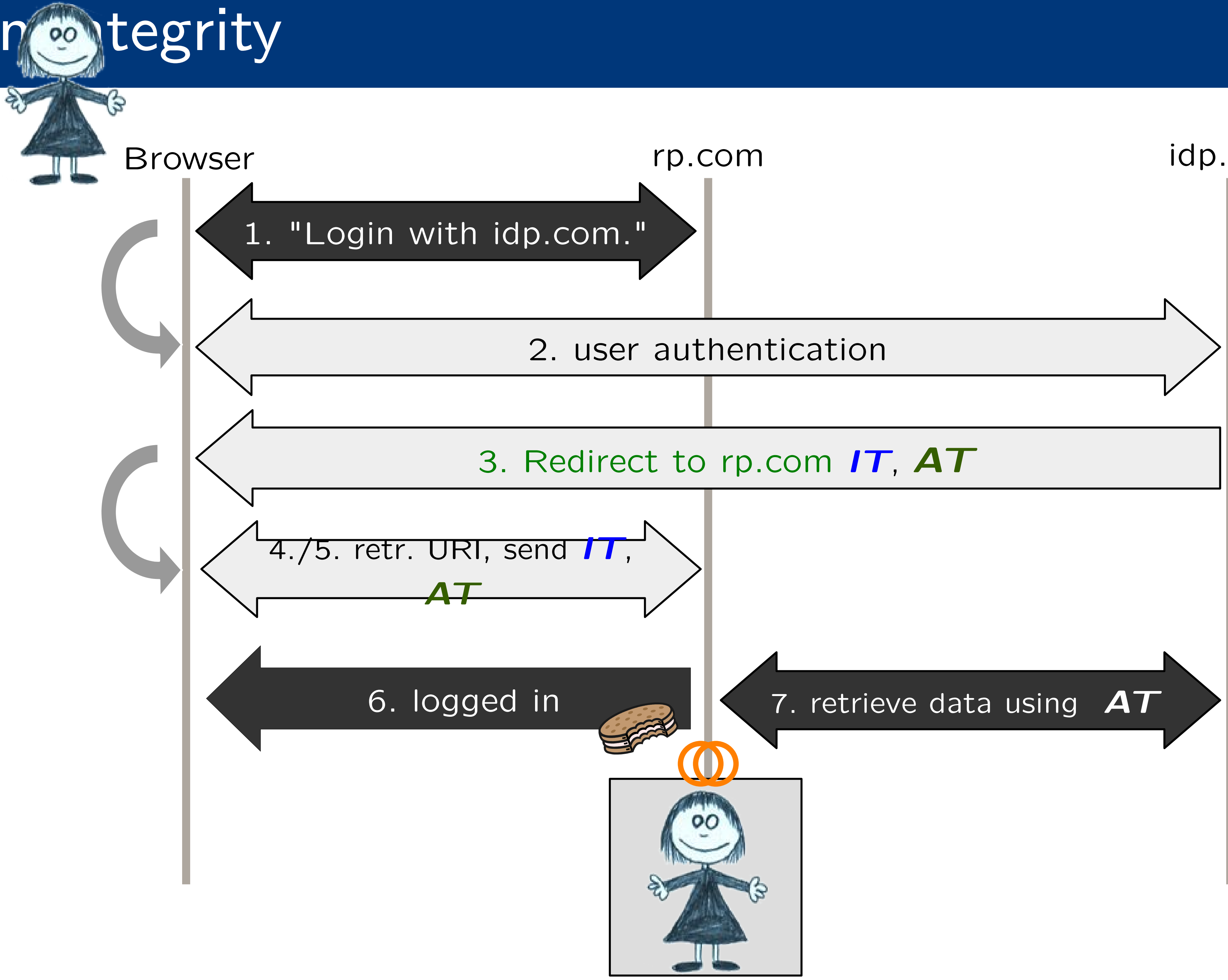
Authorization Property



Authorization Property

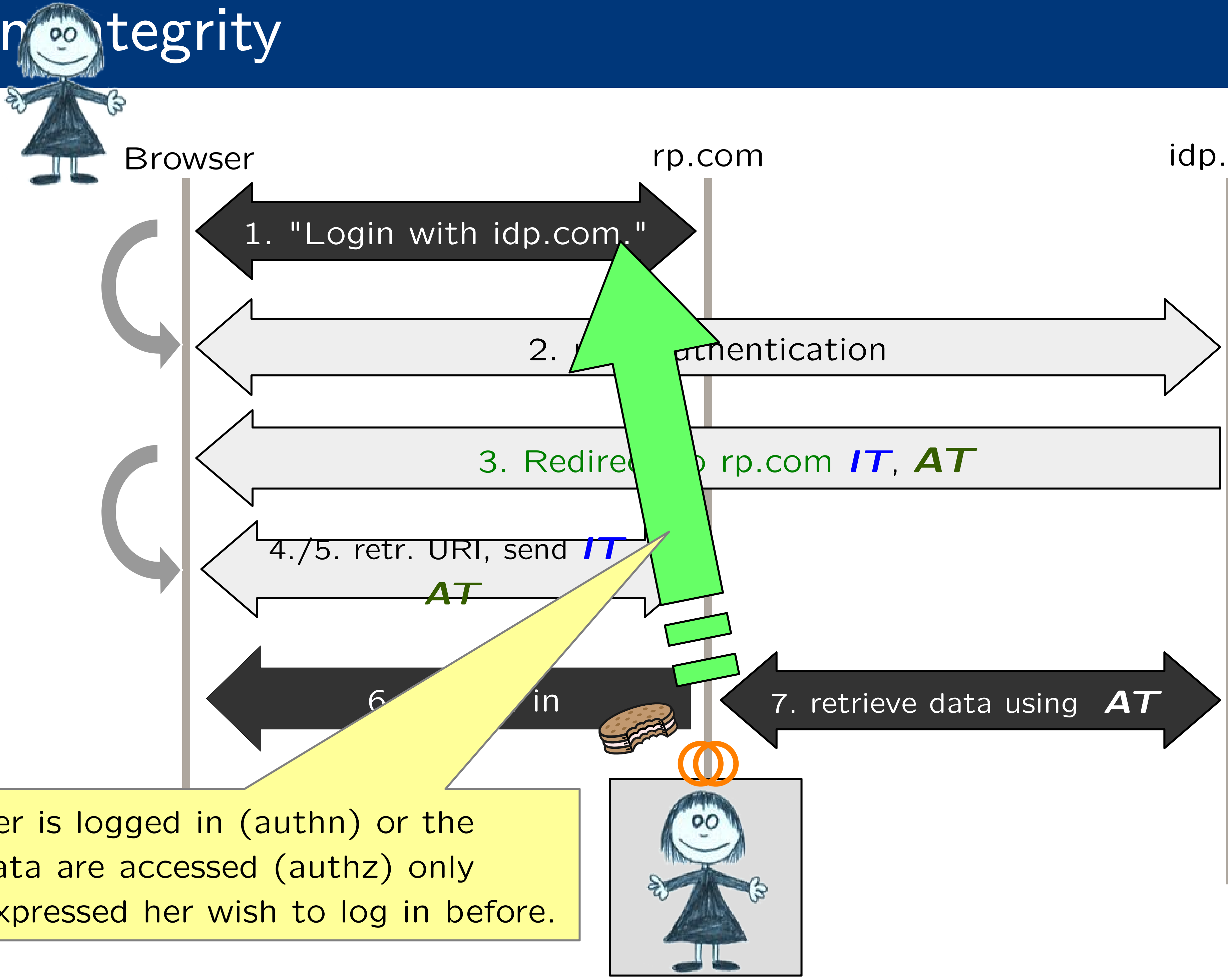


Session[illegible]tegrity



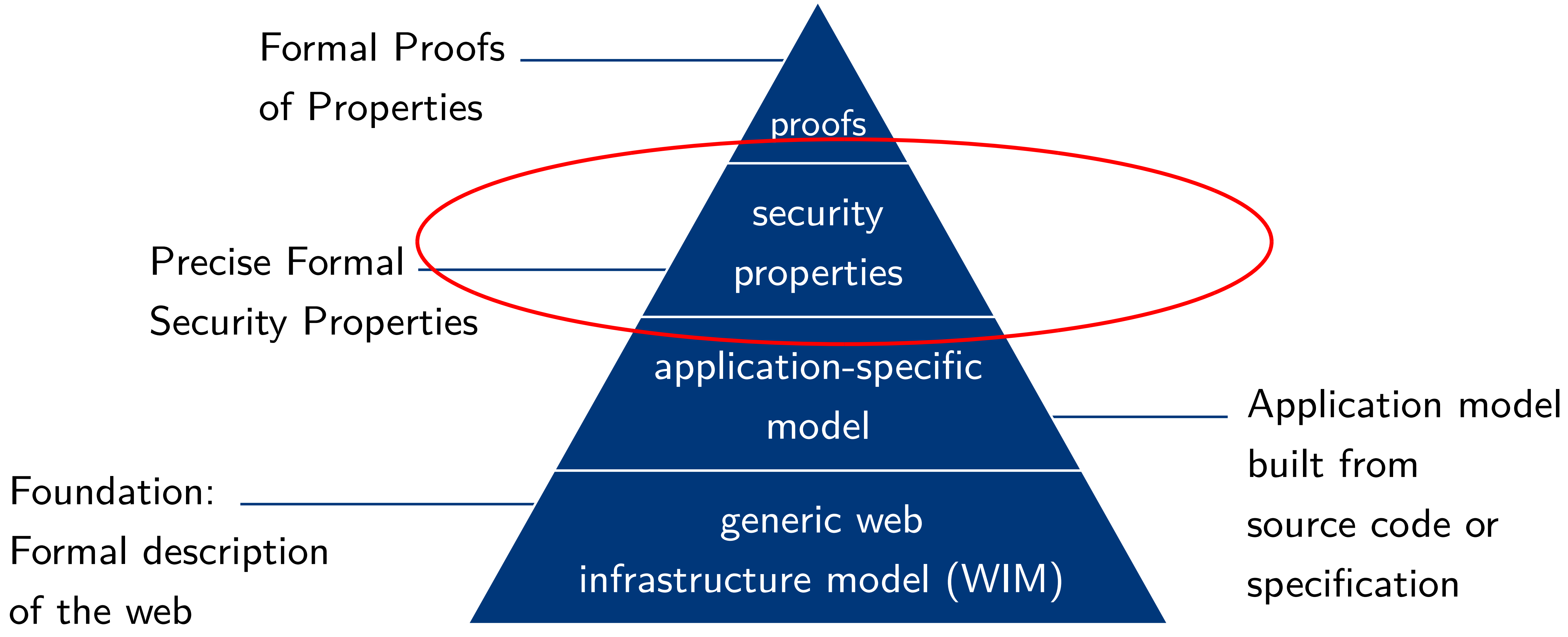
Browser
rp.com
idp.com
1. "Login with idp.com."
2. user authentication
3. Redirect to rp.com *IT*, *AT*
4./5. retr. URI, send *IT*, *AT*
6. logged in
7. retrieve data using *AT*

Session integrity

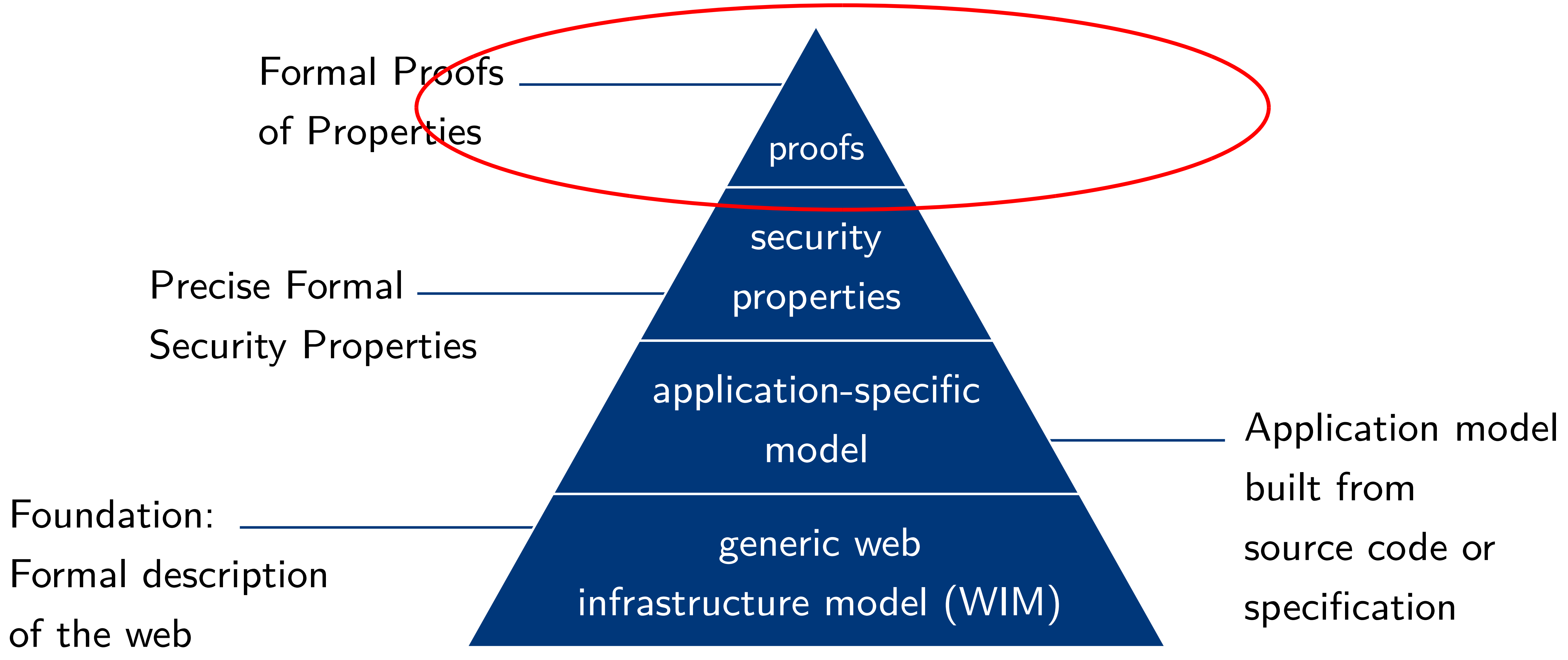


The user is logged in (authn) or the user's data are accessed (authz) only if the user expressed her wish to log in before.

How to use the WIM?


Formal Proofs of Properties
proofs
Precise Formal Security Properties
security properties
application-specific model
Application model built from source code or specification
Foundation: Formal description of the web
generic web infrastructure model (WIM)

How to use the WIM?

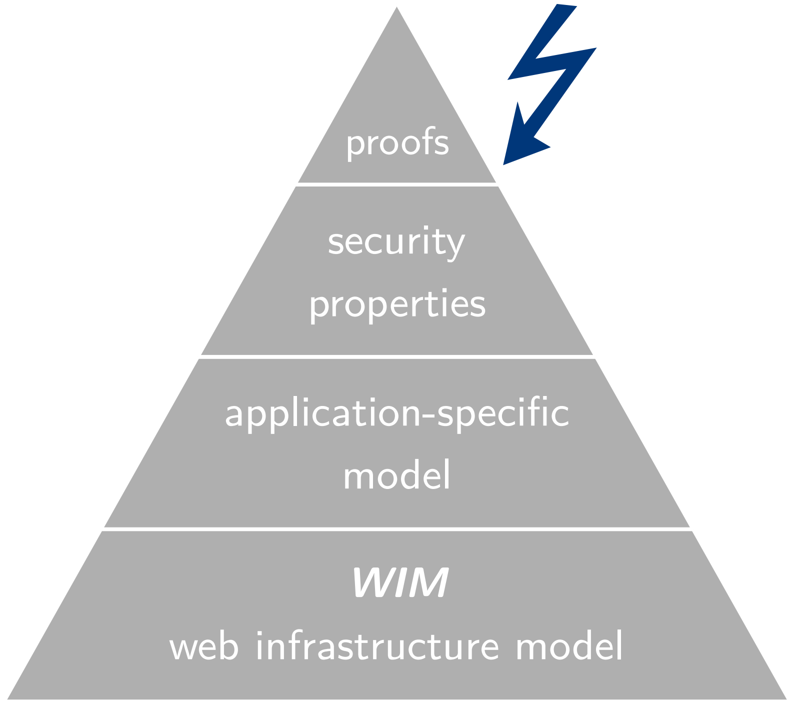

Formal Proofs of Properties
proofs
Precise Formal Security Properties
security properties
application-specific model
Application model built from source code or specification
Foundation: Formal description of the web
generic web infrastructure model (WIM)

OAuth 2.0: New Attacks

OAuth 2.0 had been analyzed many times before,
but not in a comprehensive formal model.

New attacks:

- 307 Redirect Attack
- Identity Provider Mix-Up Attack (new class of attacks)
- State Leak Attack
- Naïve Client Session Integrity Attack
- Across Identity Provider State Reuse Attack



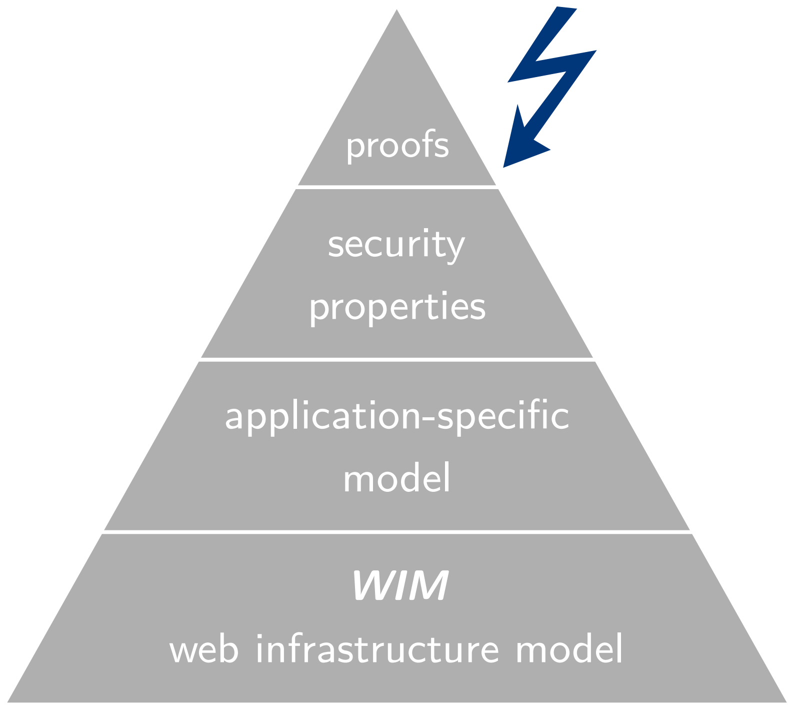
OAuth 2.0: New Attacks

OAuth 2.0 had been analyzed many times before, but not in a comprehensive formal model.

New attacks:

- 307 Redirect Attack
- Identity Provider Mix-Up Attack (new class of attacks)
- State Leak Attack
- Naïve Client Session Integrity Attack
- Across Identity Provider State Reuse Attack

Similary for OpenID Connect.



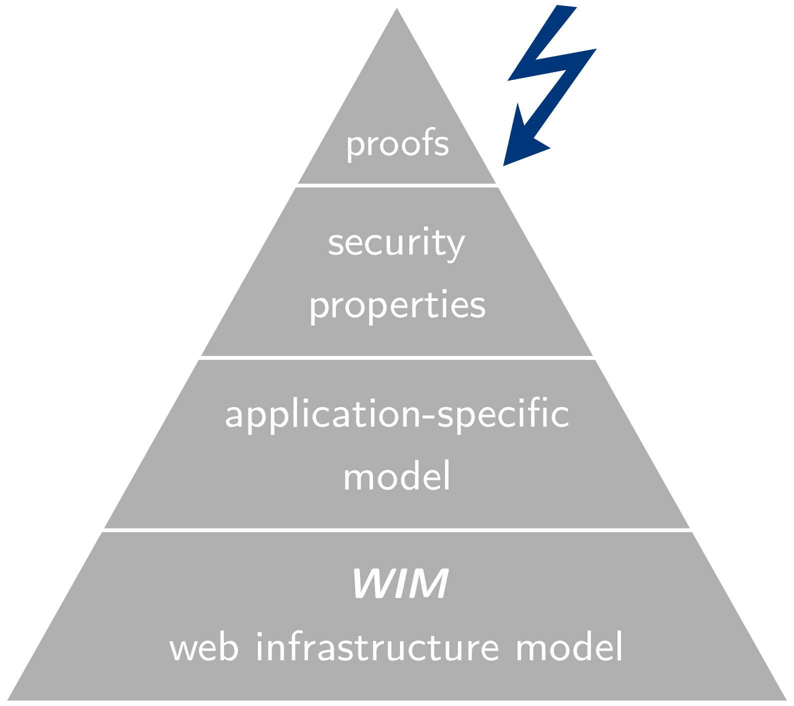
OAuth 2.0: New Attacks

OAuth 2.0 had been analyzed many times before,
but not in a comprehensive formal model.

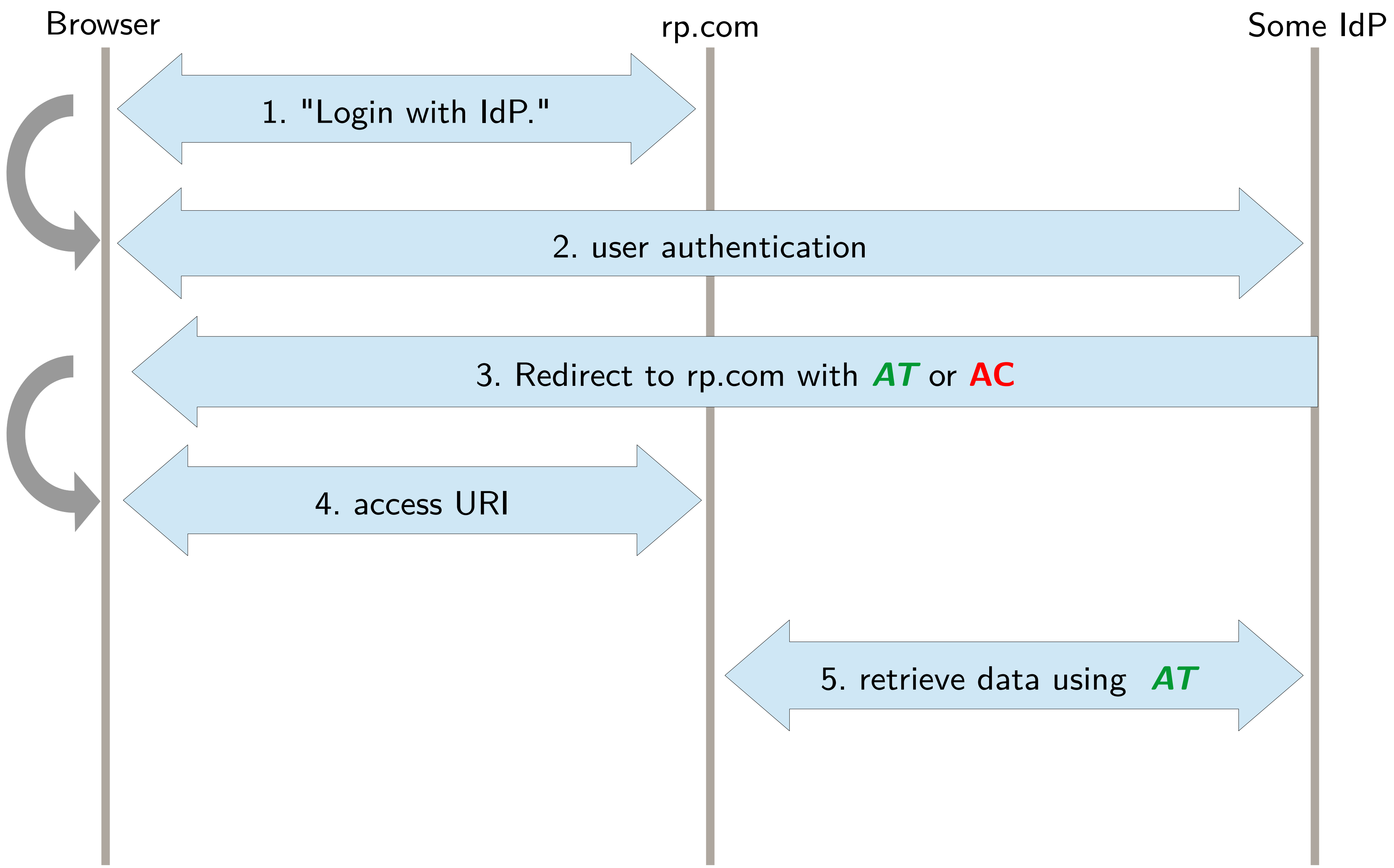
New attacks:

- 307 Redirect Attack
- Identity Provider Mix-Up Attack (new class of attacks)
- State Leak Attack
- Naïve Client Session Integrity Attack
- Across Identity Provider State Reuse Attack

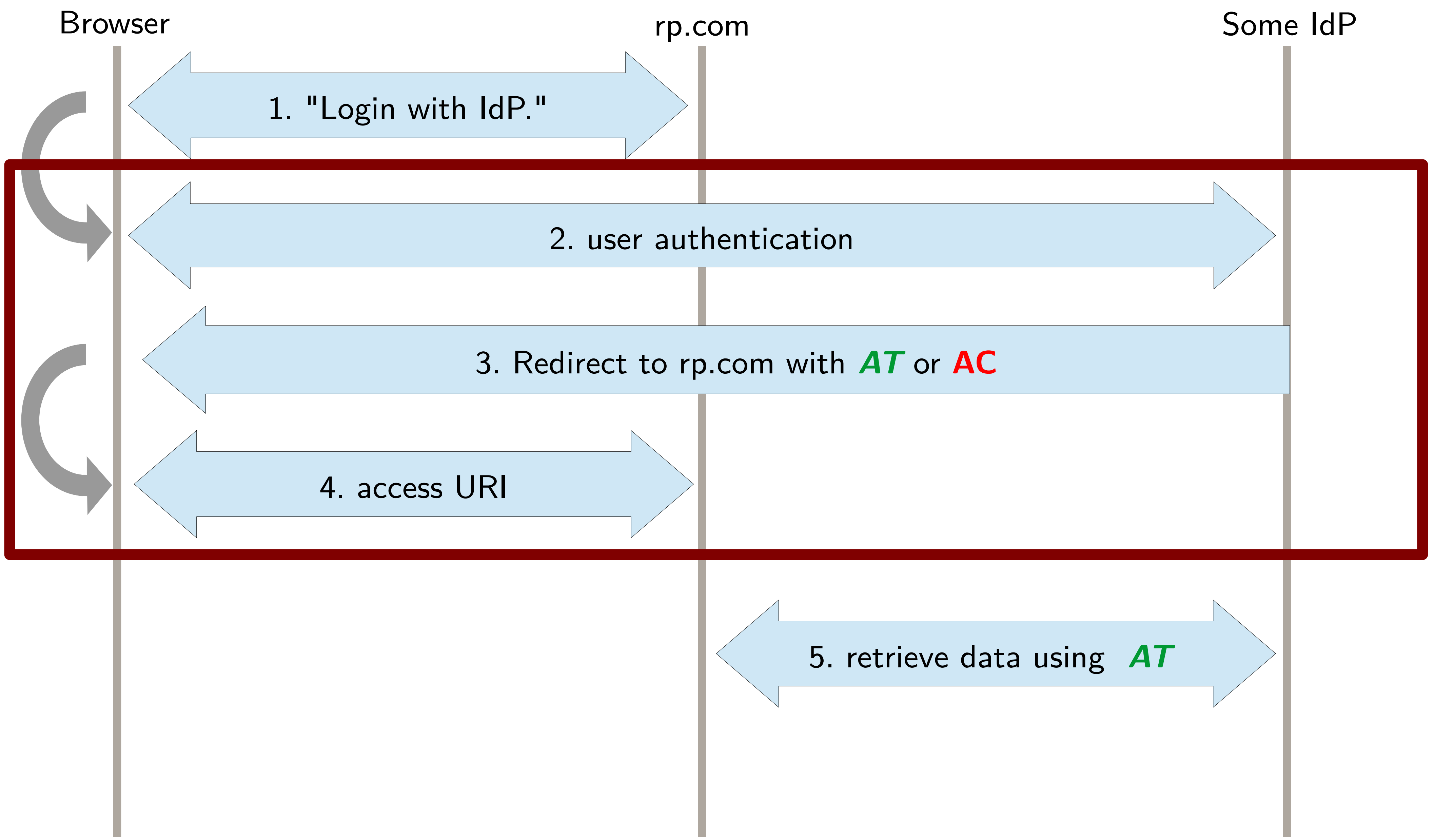
Similary for OpenID Connect.



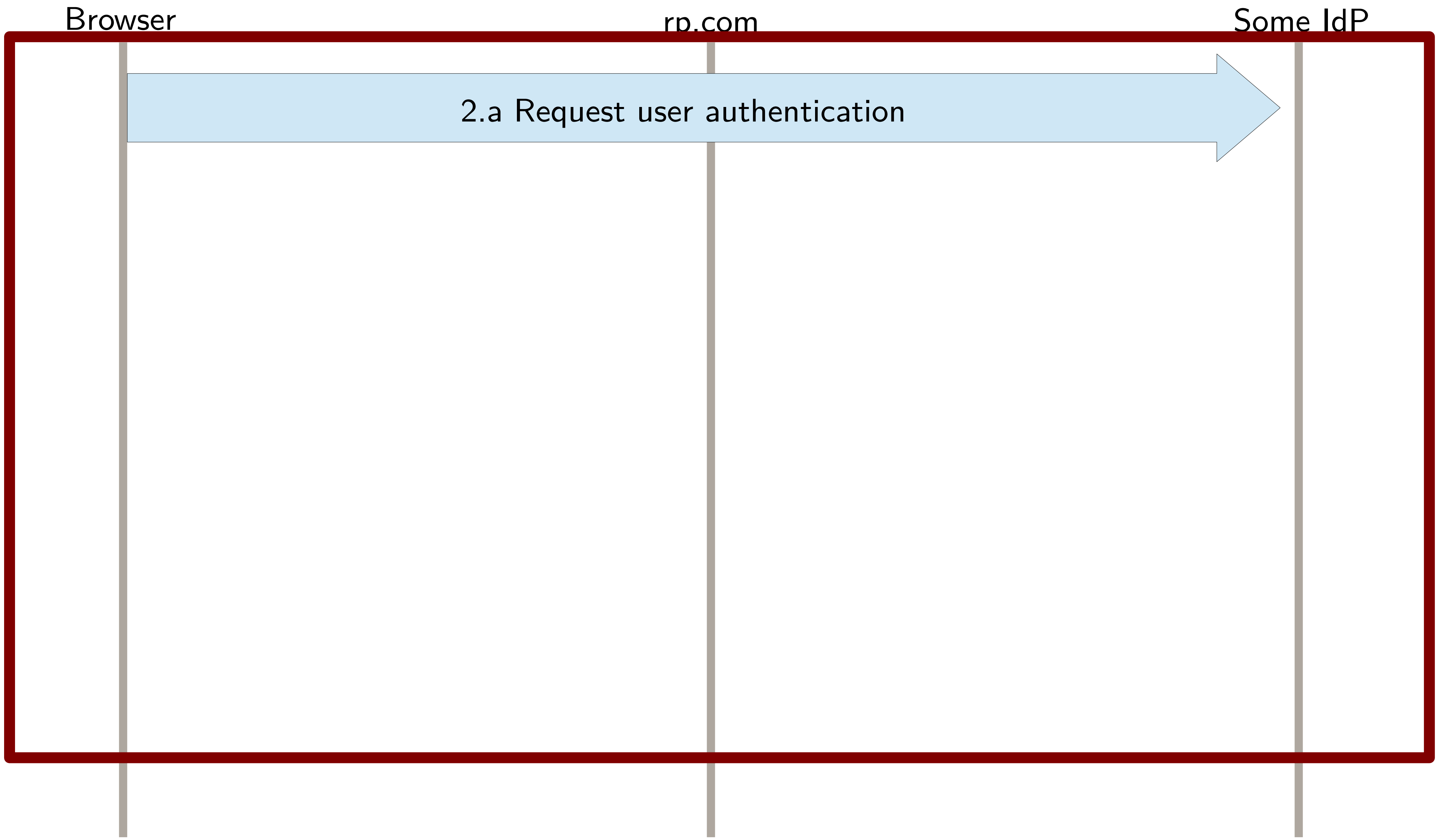
307 Redirect Attack



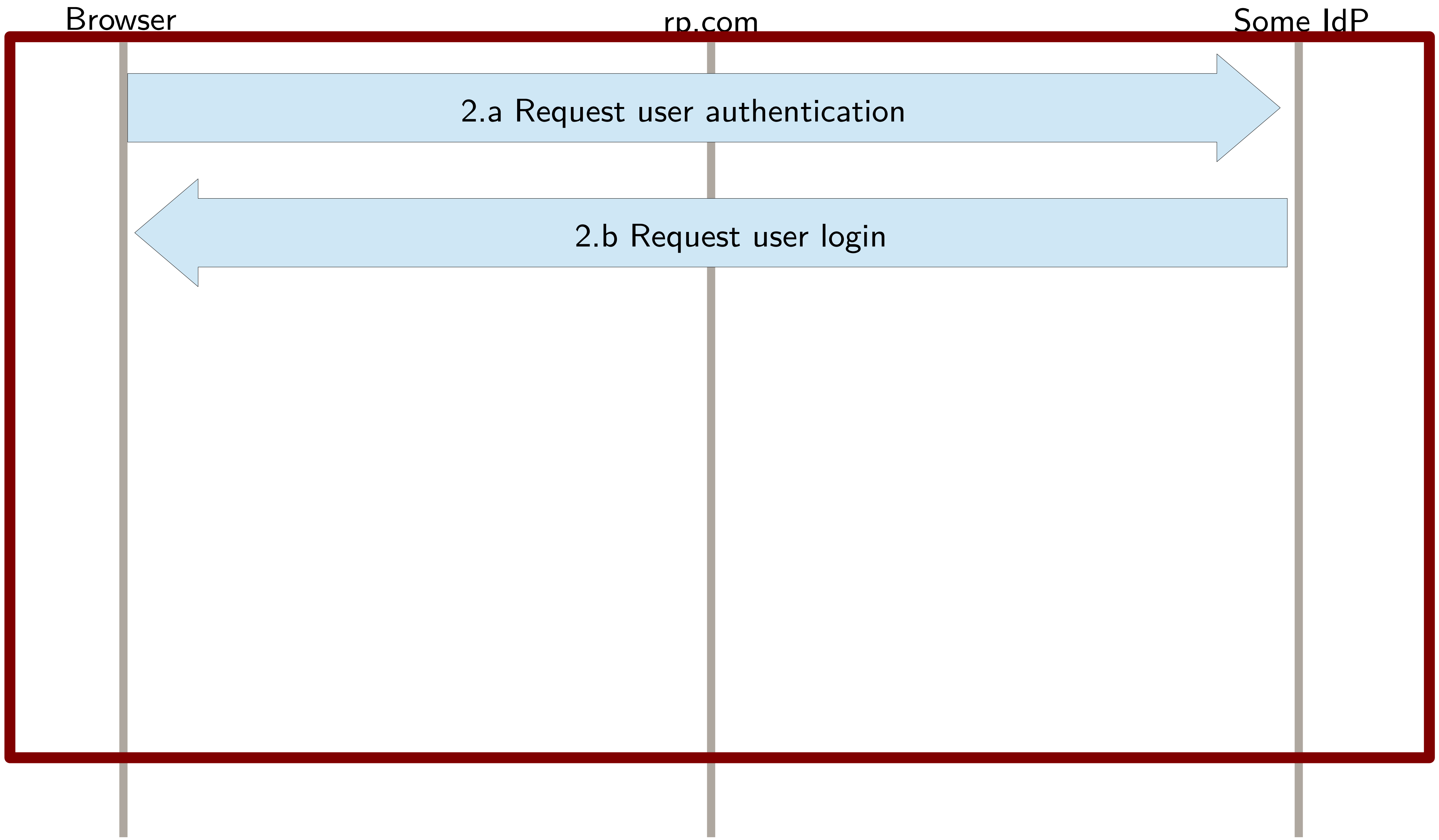
307 Redirect Attack



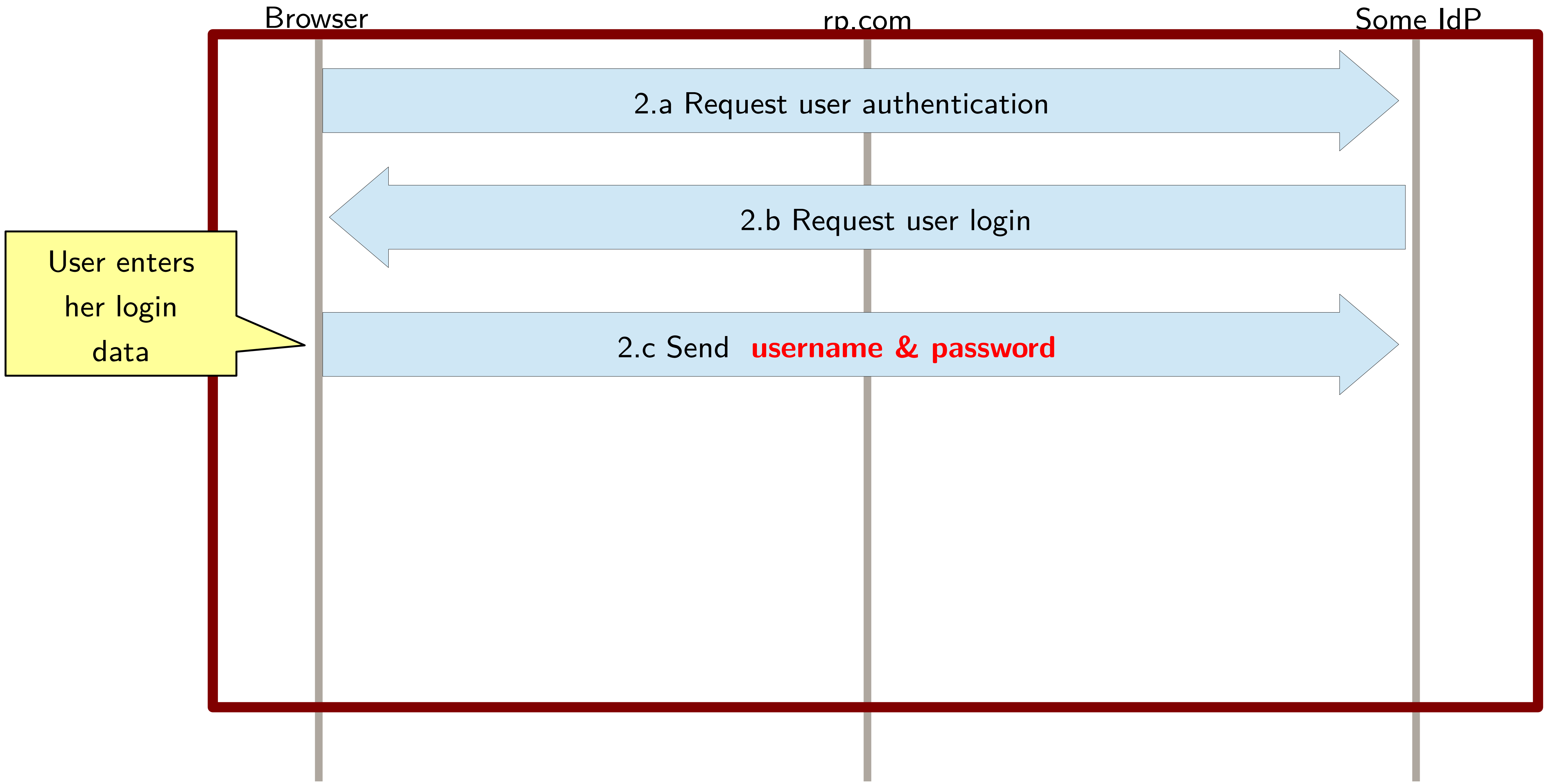
307 Redirect Attack



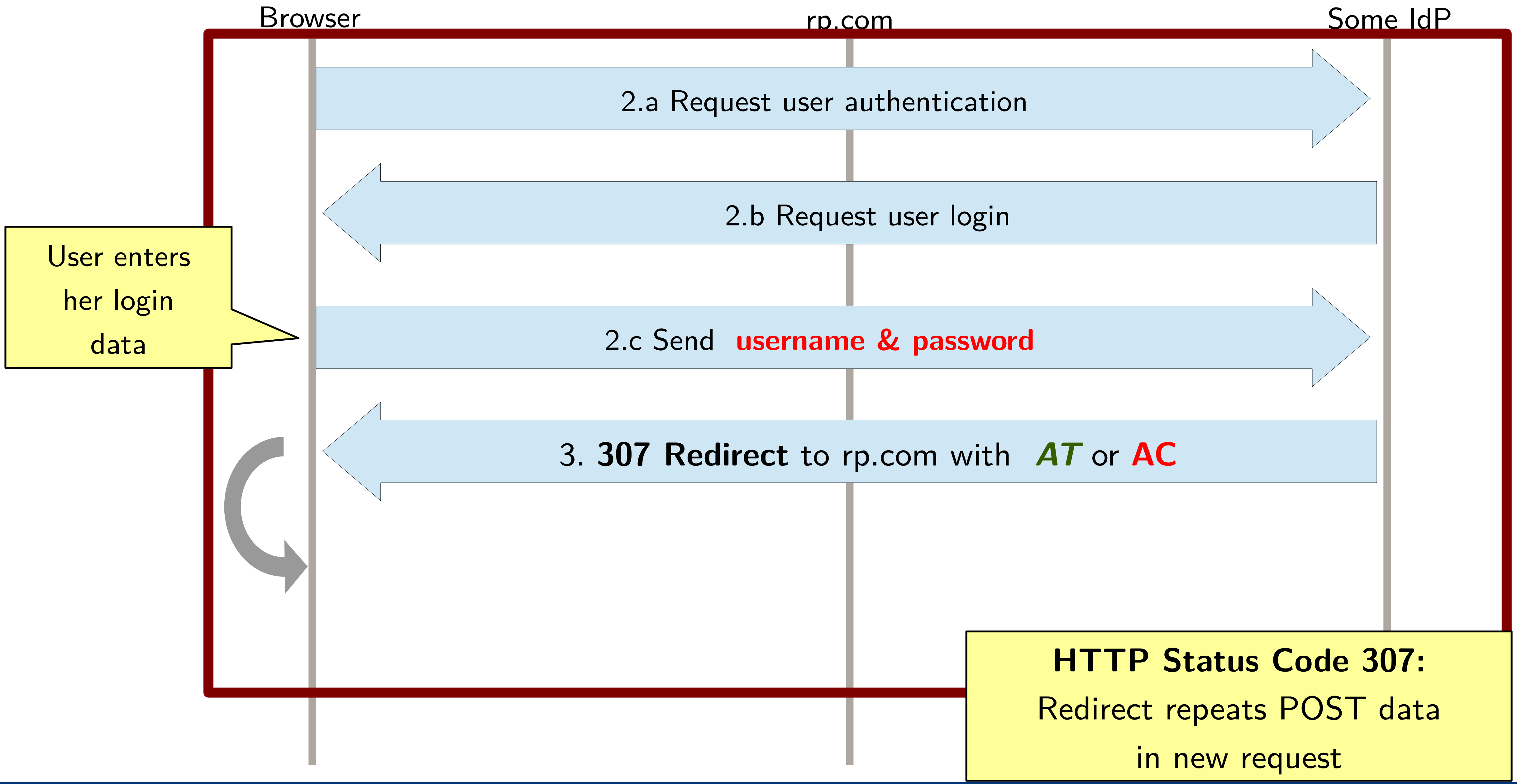
307 Redirect Attack



307 Redirect Attack

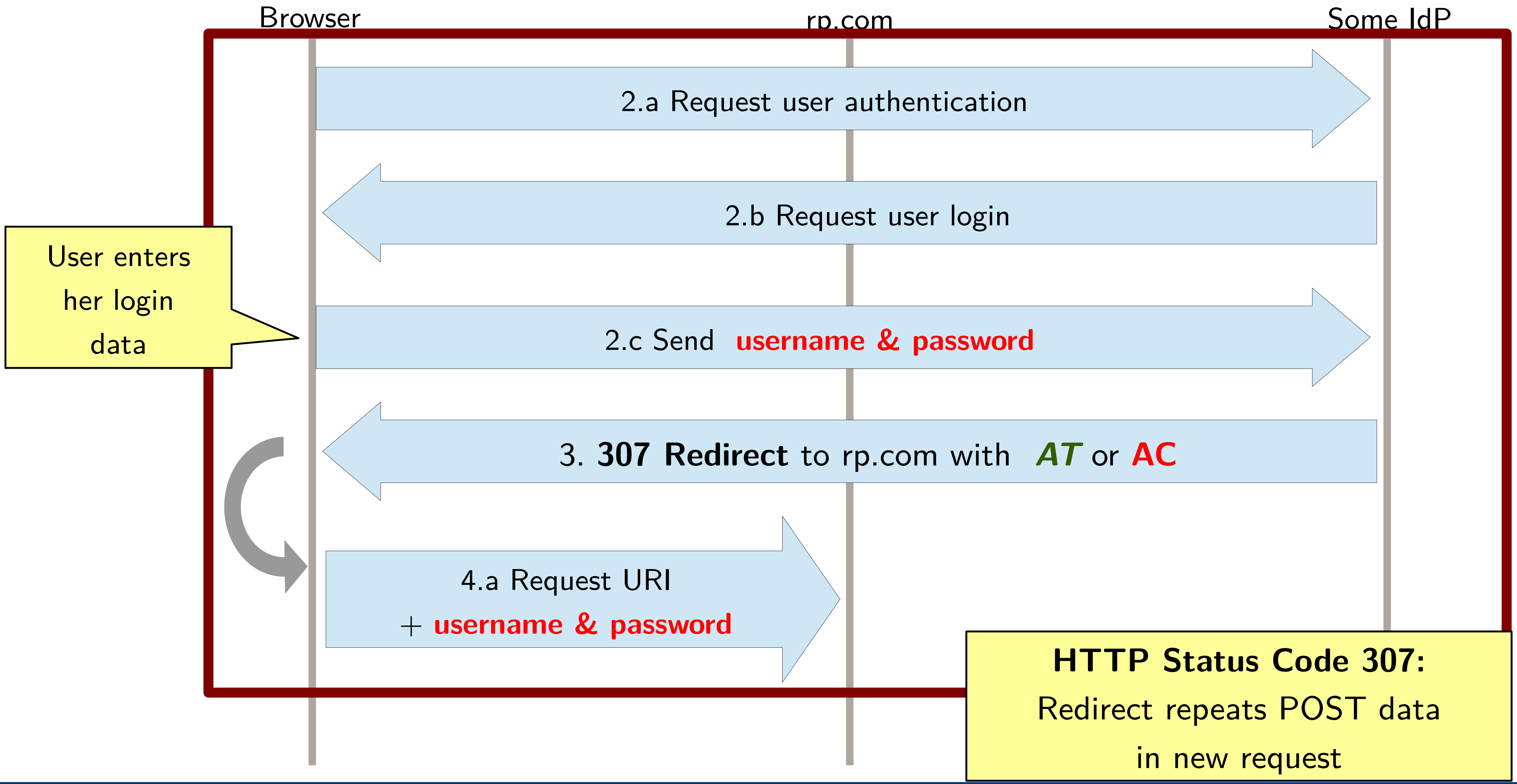

Browser
rp.com
Some IdP
2.a Request user authentication
2.b Request user login
2.c Send username & password
User enters her login data

307 Redirect Attack


Browser
rp.com
Some IdP
2.a Request user authentication
2.b Request user login
User enters her login data
2.c Send username & password
3. 307 Redirect to rp.com with AT or AC

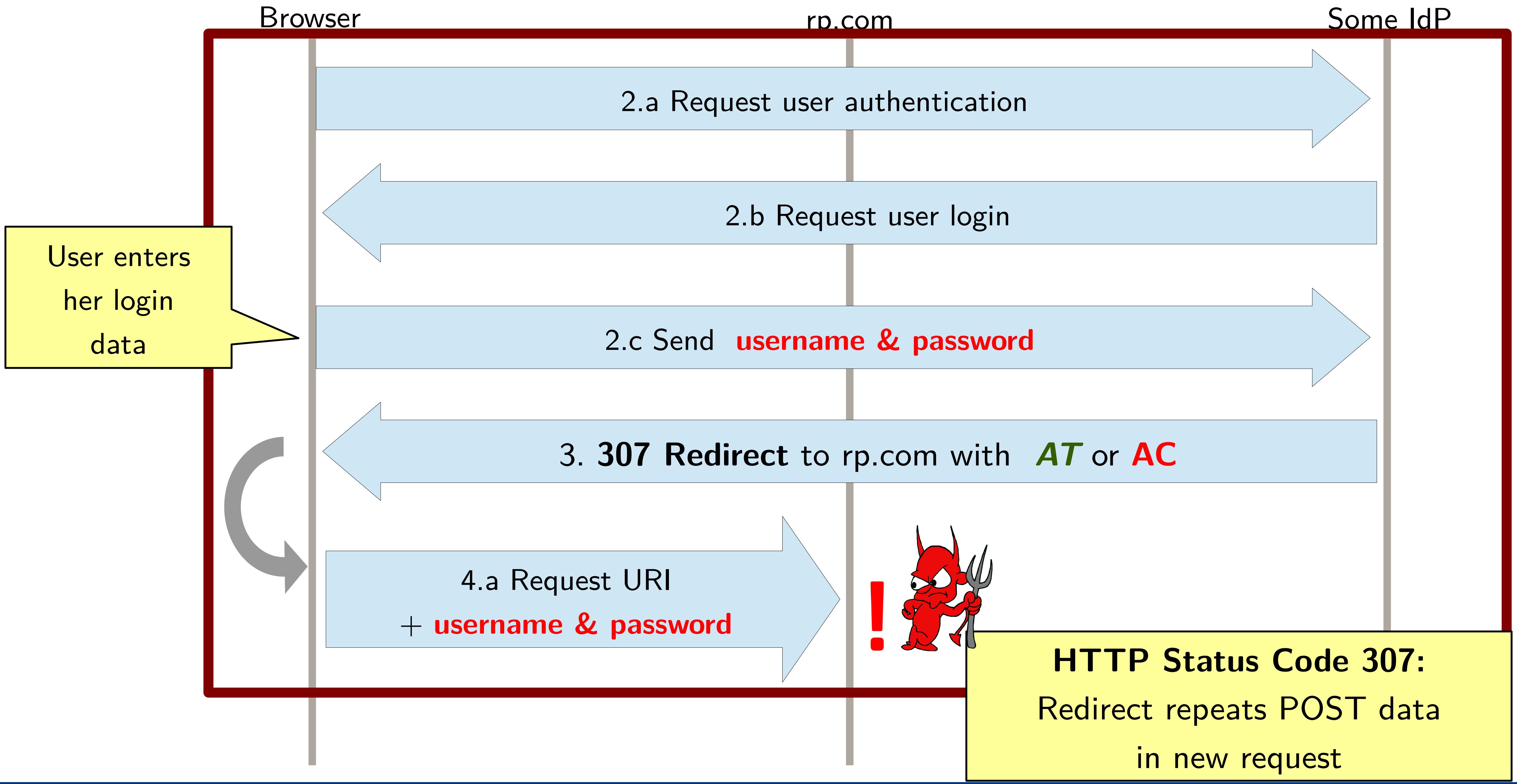
HTTP Status Code 307:
Redirect repeats POST data in new request

307 Redirect Attack



Browser
rp.com
Some IdP
2.a Request user authentication
2.b Request user login
User enters her login data
2.c Send **username & password**
3. **307 Redirect** to rp.com with ***AT*** or **AC**
4.a Request URI + **username & password**
HTTP Status Code 307: Redirect repeats POST data in new request

307 Redirect Attack

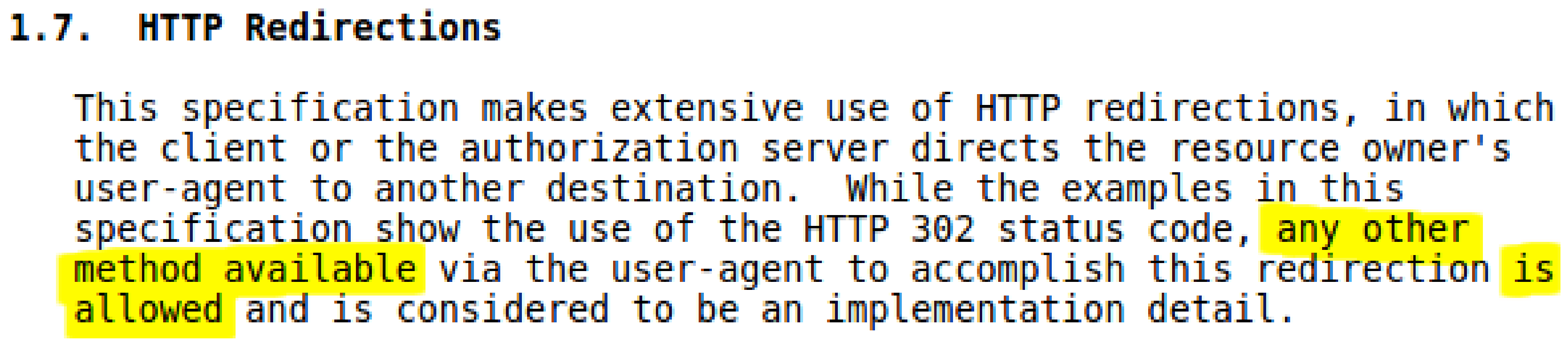

Browser
rp.com
Some IdP
2.a Request user authentication
2.b Request user login
User enters her login data
2.c Send username & password
3. 307 Redirect to rp.com with AT or AC
4.a Request URI + username & password
!

HTTP Status Code 307:
Redirect repeats POST data in new request

307 Redirect Attack

The attacker receives the username and password of the user.

OAuth standard says:



```
1.7.  HTTP Redirections

   This specification makes extensive use of HTTP redirections, in which
   the client or the authorization server directs the resource owner's
   user-agent to another destination.  While the examples in this
   specification show the use of the HTTP 302 status code, any other
   method available via the user-agent to accomplish this redirection is
   allowed and is considered to be an implementation detail.
```

Mitigation:

Use status code 303 or any other method that does not forward POST data.

Theorem

Theorem

We proposed fixed to the standards and proved them secure:

Theorem

OAuth 2.0 and OIDC with fixes fulfill security properties

- Authentication
- Authorization
- Session Integrity

Impact

- Disclosed OAuth 2.0 attacks to the IETF Web Authorization Working Group in late 2015 (and had emergency meeting)
- Since then: In close contact with the IETF and OpenID Foundation to improve standards
- Initiated the OAuth Security Workshop (OSW) to foster the exchange between researchers, standardization groups, and industry.
  This year in its 10th edition (OSW 2025).

More Recent Case Studies: New (High-Risk) Environments and More Functionality/Flexibility

SSO: Today

Browser

Relying Party/Client
e.g. tripadvisor.com

Identity Provider (IdP)
e.g. facebook.com

Apps

- **IoT**
  - Car
  - TV
- **Open Banking/Open Finance**
  - FinTech
- **Open Health**
  - Doctor/Pharmacy
- **Open Government**
  - Tax consultant

More secure and complex protocols needed

Assume stronger and more motivated attackers

acturer

- **Open Banking/Open Finance**
  - Bank
- **Open Health**
  - Health Insurance
  - National Health Service
- **Open Government**
  - Government/Tax/Registry

SSO: Today

Browser

Relying Party/Client
e.g. tripadvisor.com

Identity Provider (IdP)
e.g. facebook.com

Apps

- **IoT**
  - Car
  - TV
- **Open Banking/Open Finance**
  - FinTech
- **Open Health**
  - Doctor/Pharmacy
- **Open Government**
  - Tax consultant

More secure and complex protocols needed

Assume stronger and more motivated attackers

acturer

- **Open Banking/Open Finance**
  - Bank
- **Open Health**
  - Health Insurance
  - National Health Service
- **Open Government**
  - Government/Tax/Registry

SSO: Today

Browser

Relying Party/Client
e.g. tripadvisor.com

Identity Provider (IdP)
e.g. facebook.com

Apps

More secure and
complex protocols needed

Assume stronger and more
motivated attackers

- **IoT**
  - ➔ Car
  - ➔ TV
- **Open Banking/Open Finance**
  - ➔ FinTech
- **Open Health**
  - ➔ Doctor/Pharmacy
- **Open Government**
  - ➔ Tax consultant

acturer

- **Open Banking/Open Finance**
  - ➔ Bank
- **Open Health**
  - ➔ Health Insurance
  - ➔ National Health Service
- **Open Government**
  - ➔ Government/Tax/Registry

SSO: Today

Browser

Relying Party/Client
e.g. tripadvisor.com

Identity Provider (IdP)
e.g. facebook.com

Apps

FAPI 1.0 and

More secure and
complex protocols needed

Assume stronger and more
motivated attackers

- **IoT**
  - Car
  - TV
- **Open Banking/Open Finance**
  - FinTech
- **Open Health**
  - Doctor/Pharmacy
- **Open Government**
  - Tax consultant

acturer

- **Open Banking/Open Finance**
  - Bank
- **Open Health**
  - Health Insurance
  - National Health Service
- **Open Government**
  - Government/Tax/Registry

SSO: Today

Browser

Relying Party/Client
e.g. tripadvisor.com

Identity Provider (IdP)
e.g. facebook.com

Apps

- **IoT**
  - ➔ Car
  - ➔ TV
- **Open Banking/Open Finance**
  - ➔ FinTech
- **Open Health**
  - ➔ Doctor/Pharmacy
- **Open Government**
  - ➔ Tax consultant

More secure and
complex protocols needed

Assume stronger and more
motivated attackers

acturer

- **Open Banking/Open Finance**
  - ➔ Bank
- **Open Health**
  - ➔ Health Insurance
  - ➔ National Health Service
- **Open Government**
  - ➔ Government/Tax/Registry

FAPI 1.0 and
FAPI 2.0

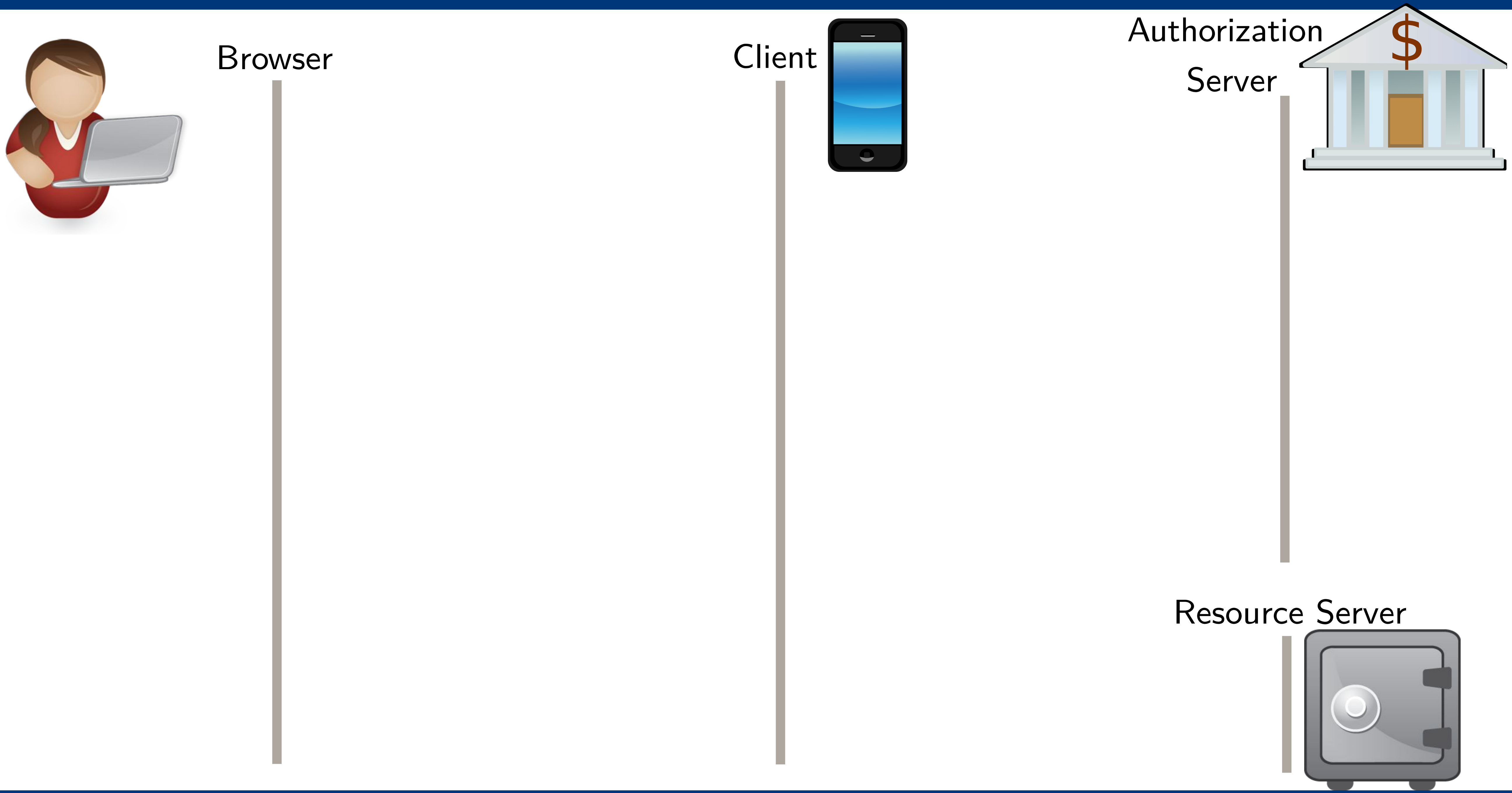
Background: FAPI

- Open Banking UK
- Open Banking Brazil
- Open Insurance Brazil
- Open Finance Brazil
- Australia's Consumer Data Standards
- Open Banking Saudi Arabia
- Financial Data Exchange
- New Zealand's core payment clearing house payments.nz
- Norway's national health data sharing

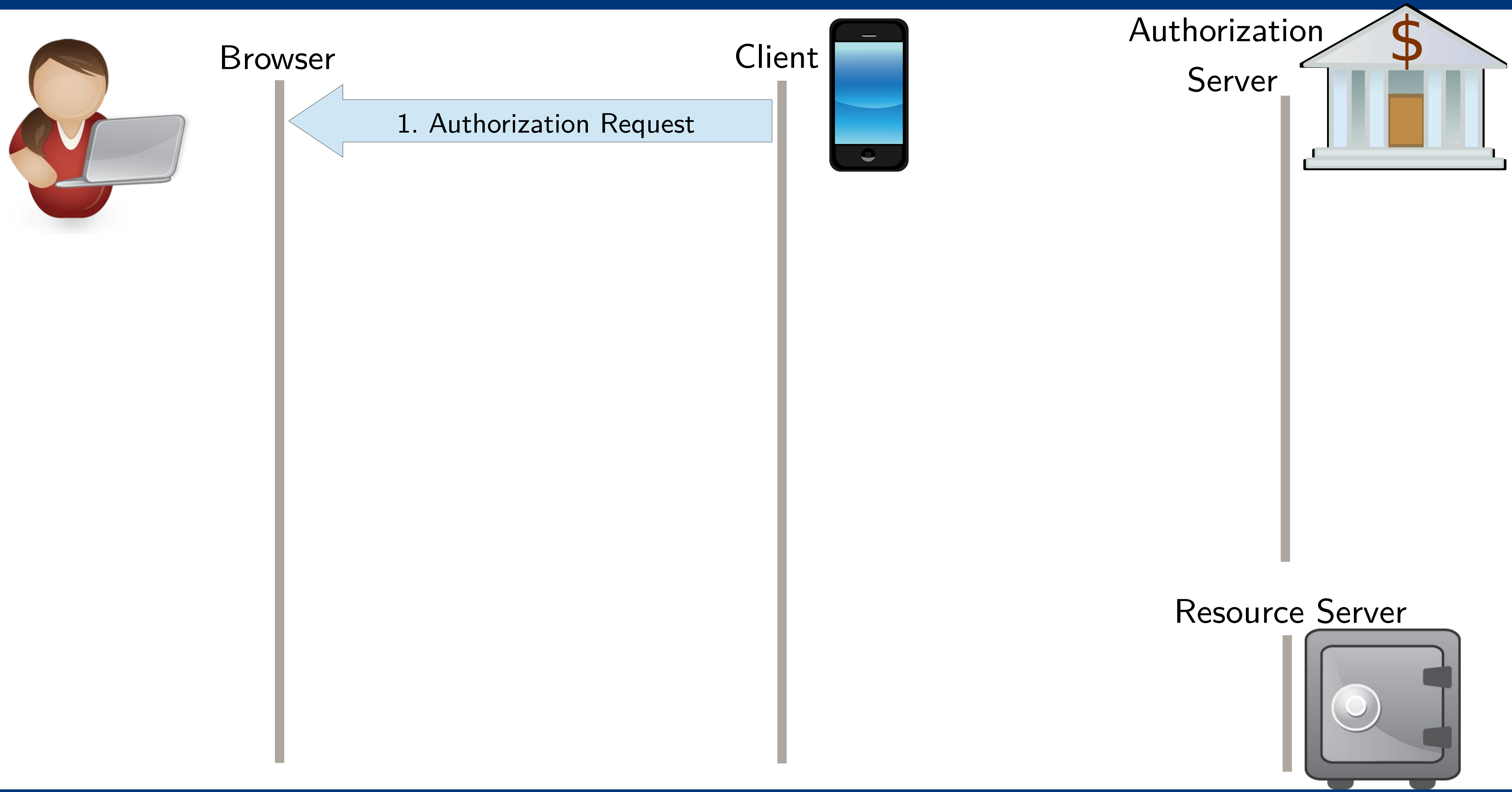
⇒ **Many millions of users in high-risk environments**

OAuth 2.0: Authorization Code Mode

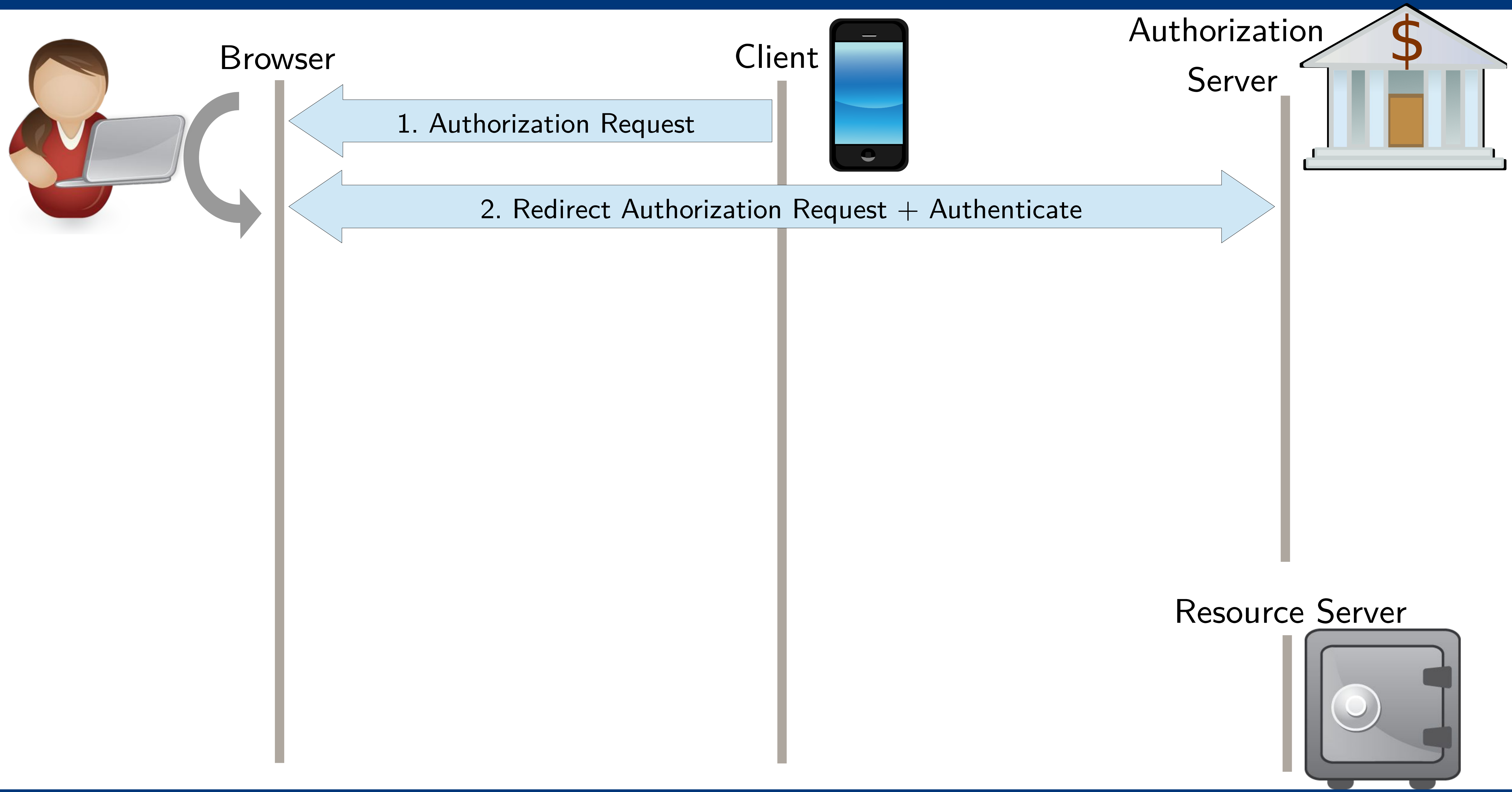
OAuth 2.0: Authorization Code Mode



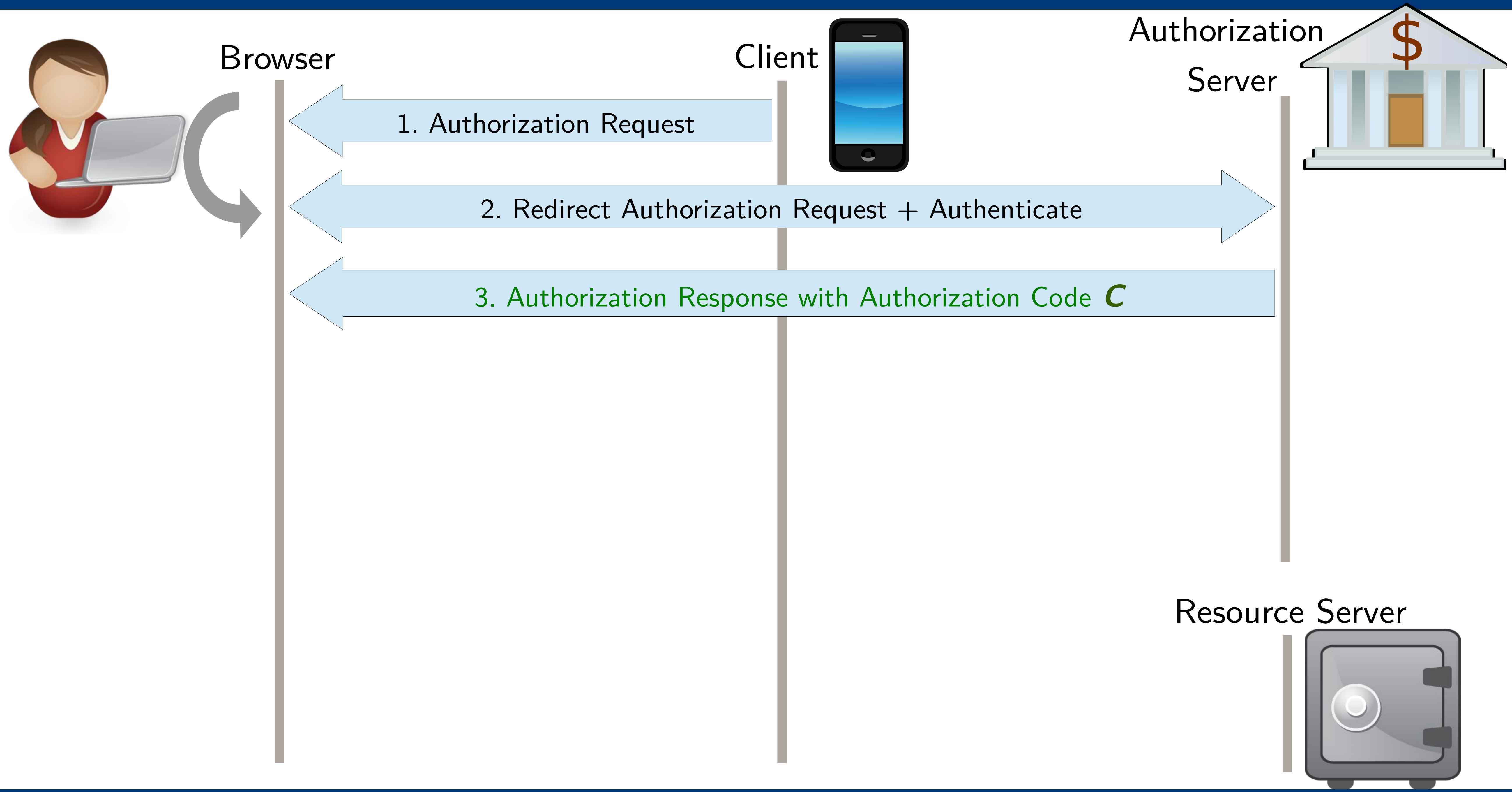
OAuth 2.0: Authorization Code Mode



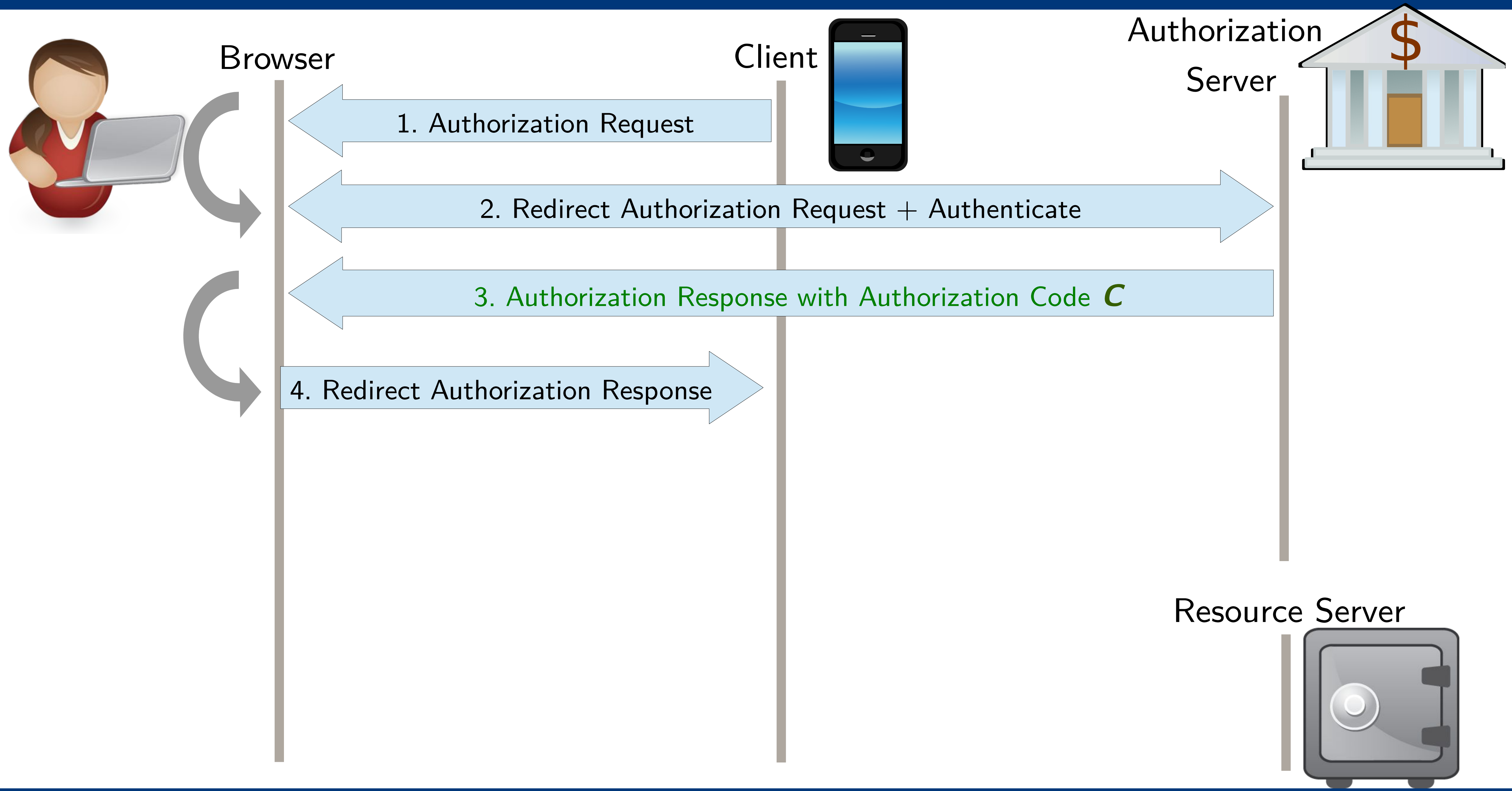
OAuth 2.0: Authorization Code Mode



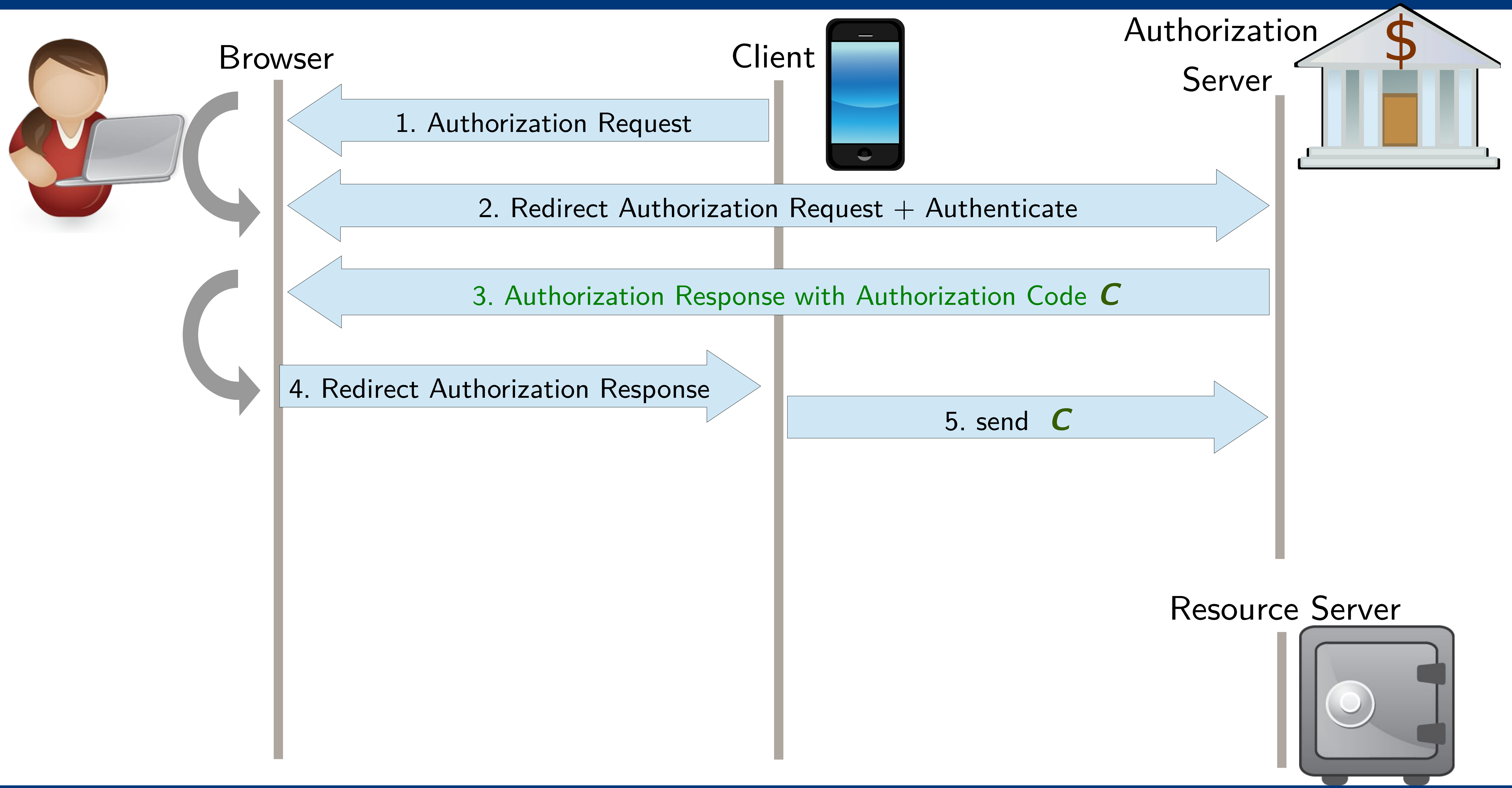
OAuth 2.0: Authorization Code Mode


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
Resource Server

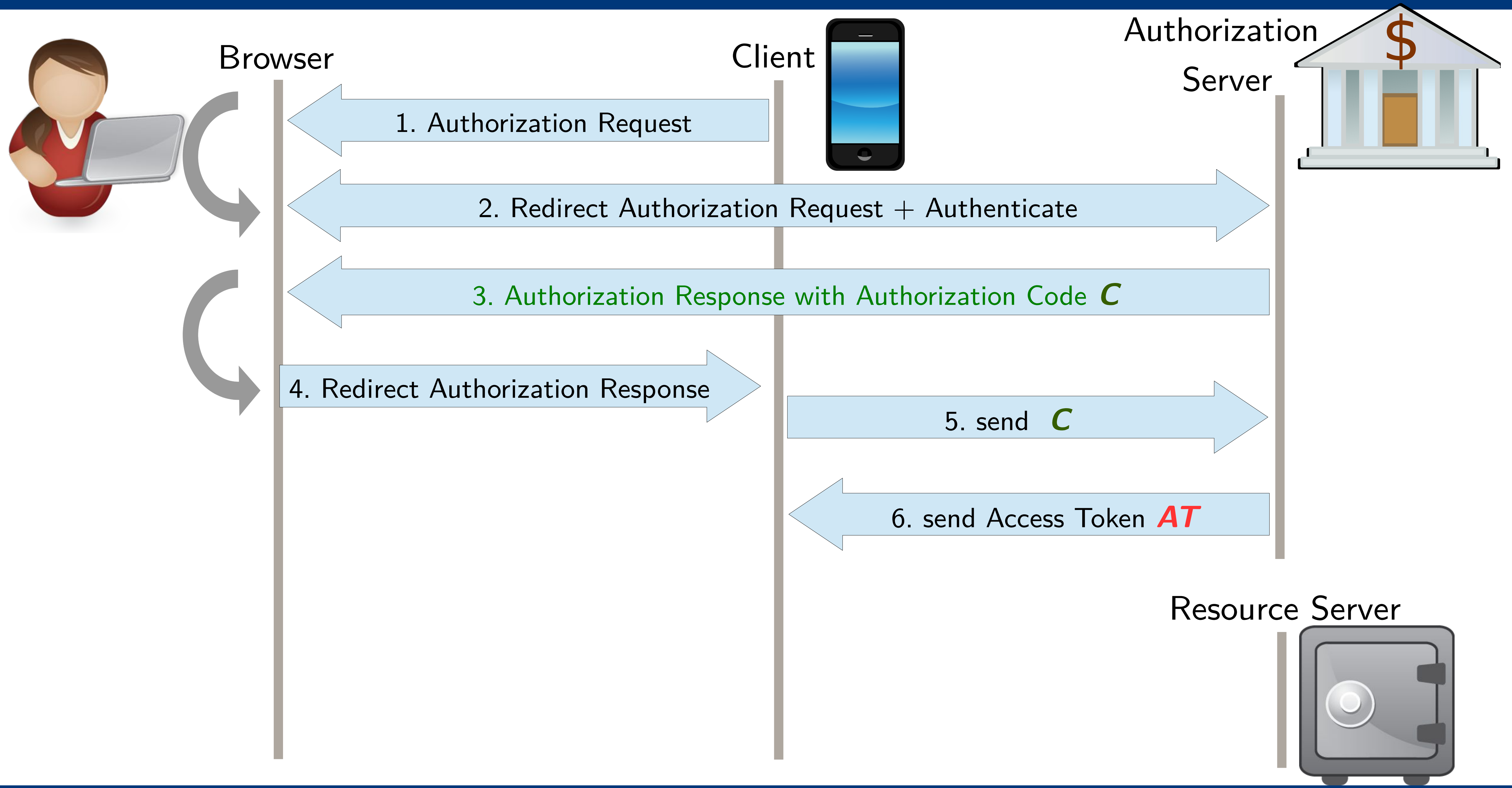
OAuth 2.0: Authorization Code Mode


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code ***C***
4. Redirect Authorization Response
Resource Server

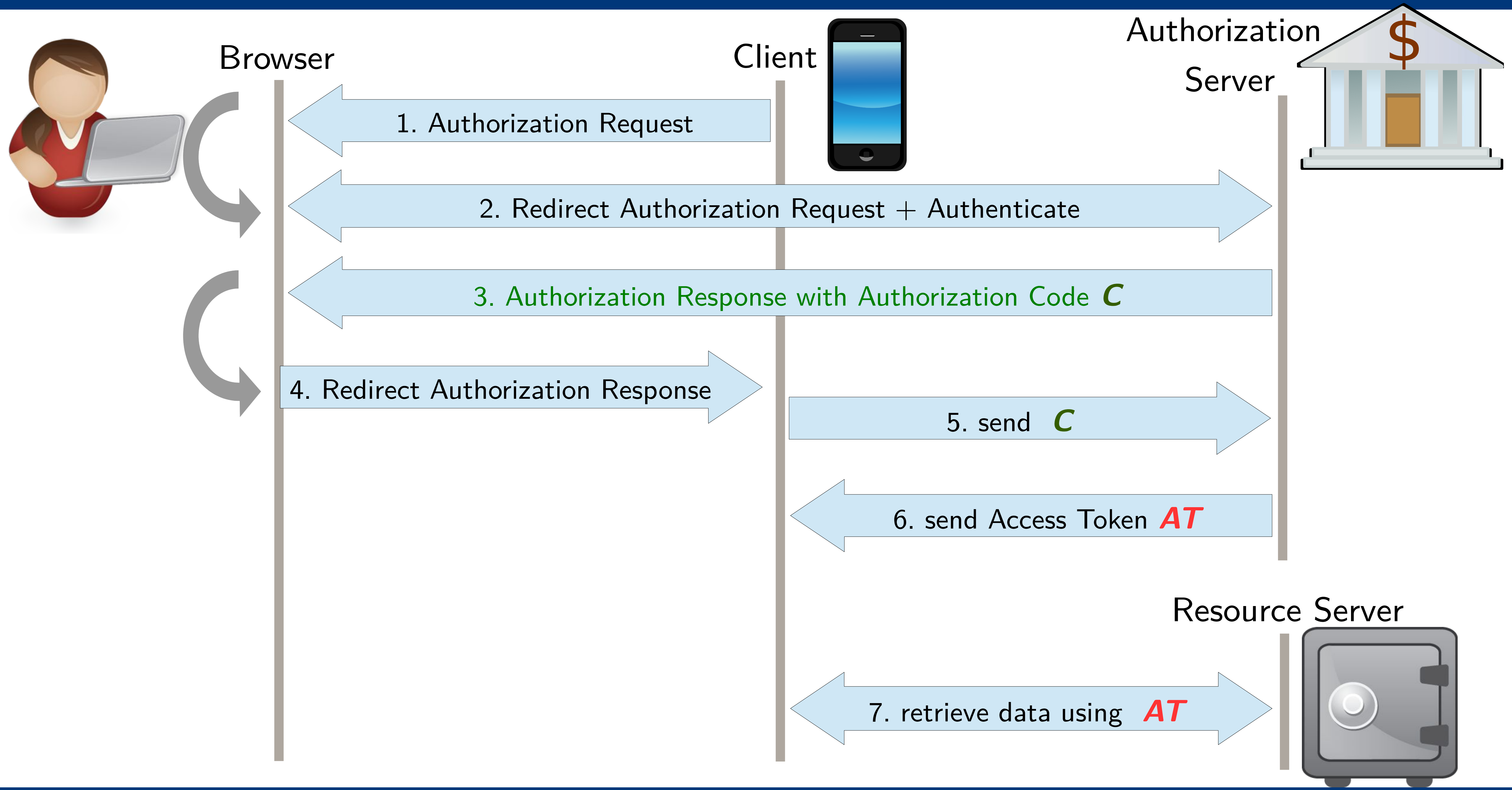
OAuth 2.0: Authorization Code Mode


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
Resource Server

OAuth 2.0: Authorization Code Mode

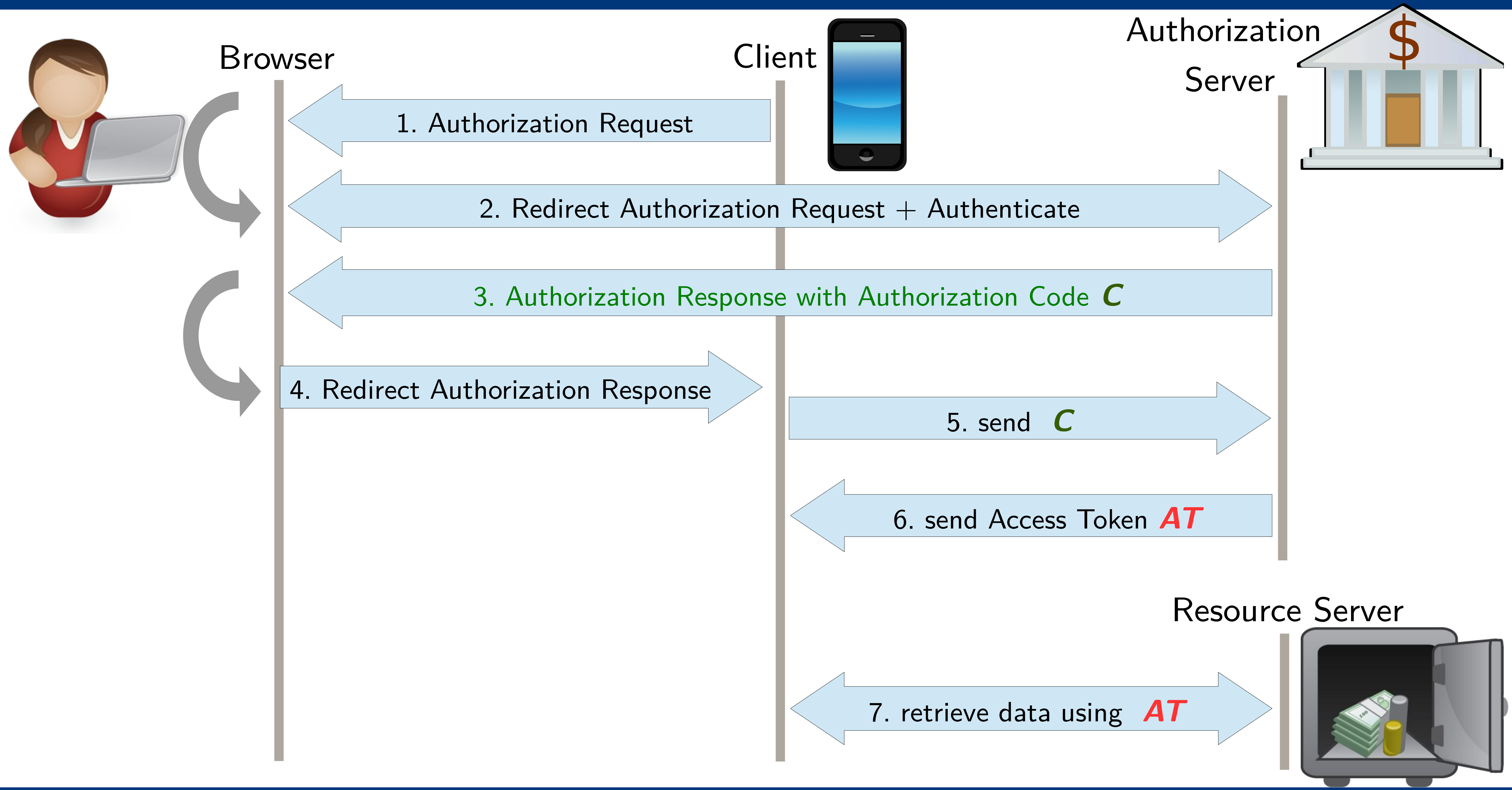

Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server

OAuth 2.0: Authorization Code Mode

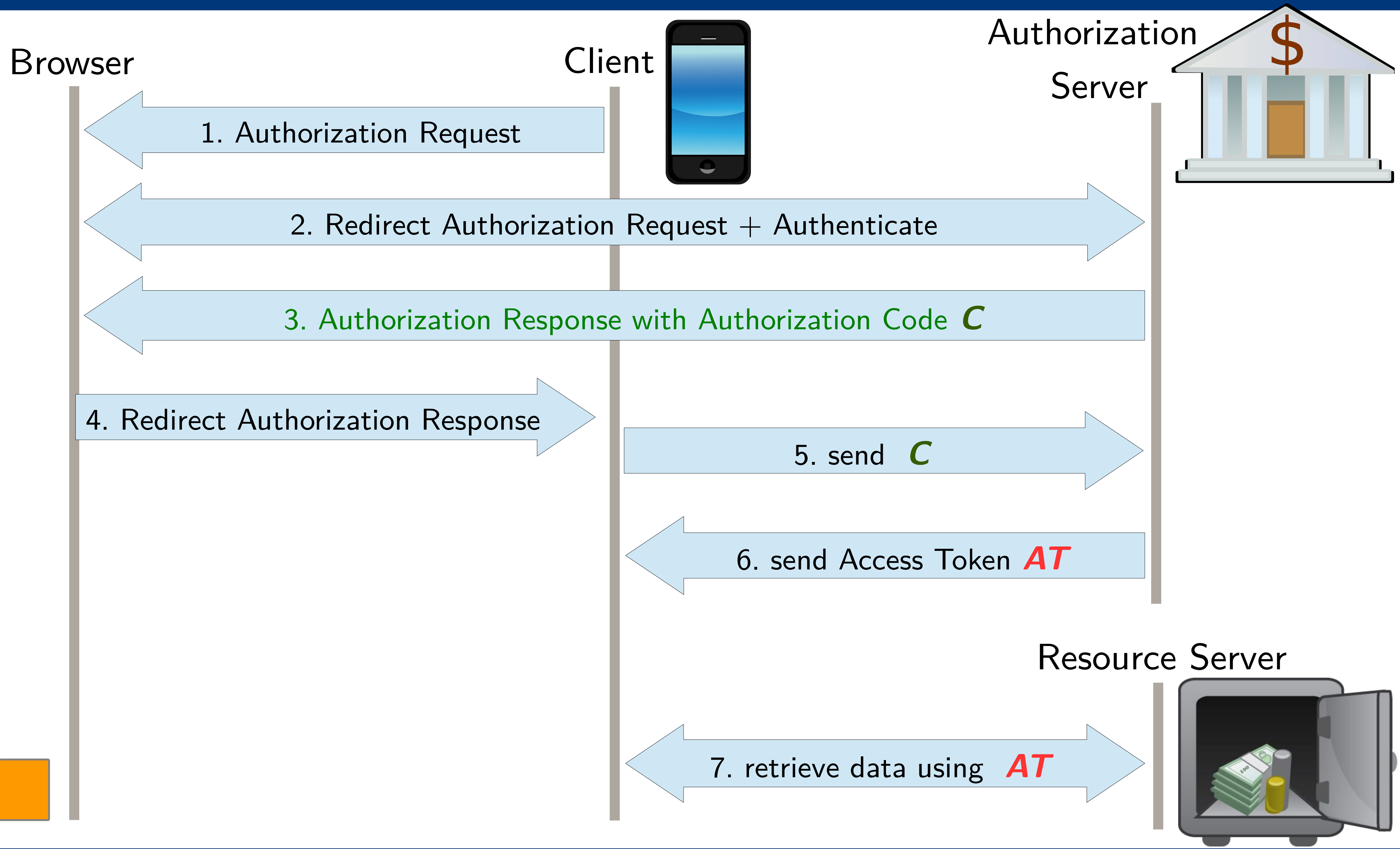


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

OAuth 2.0: Authorization Code Mode

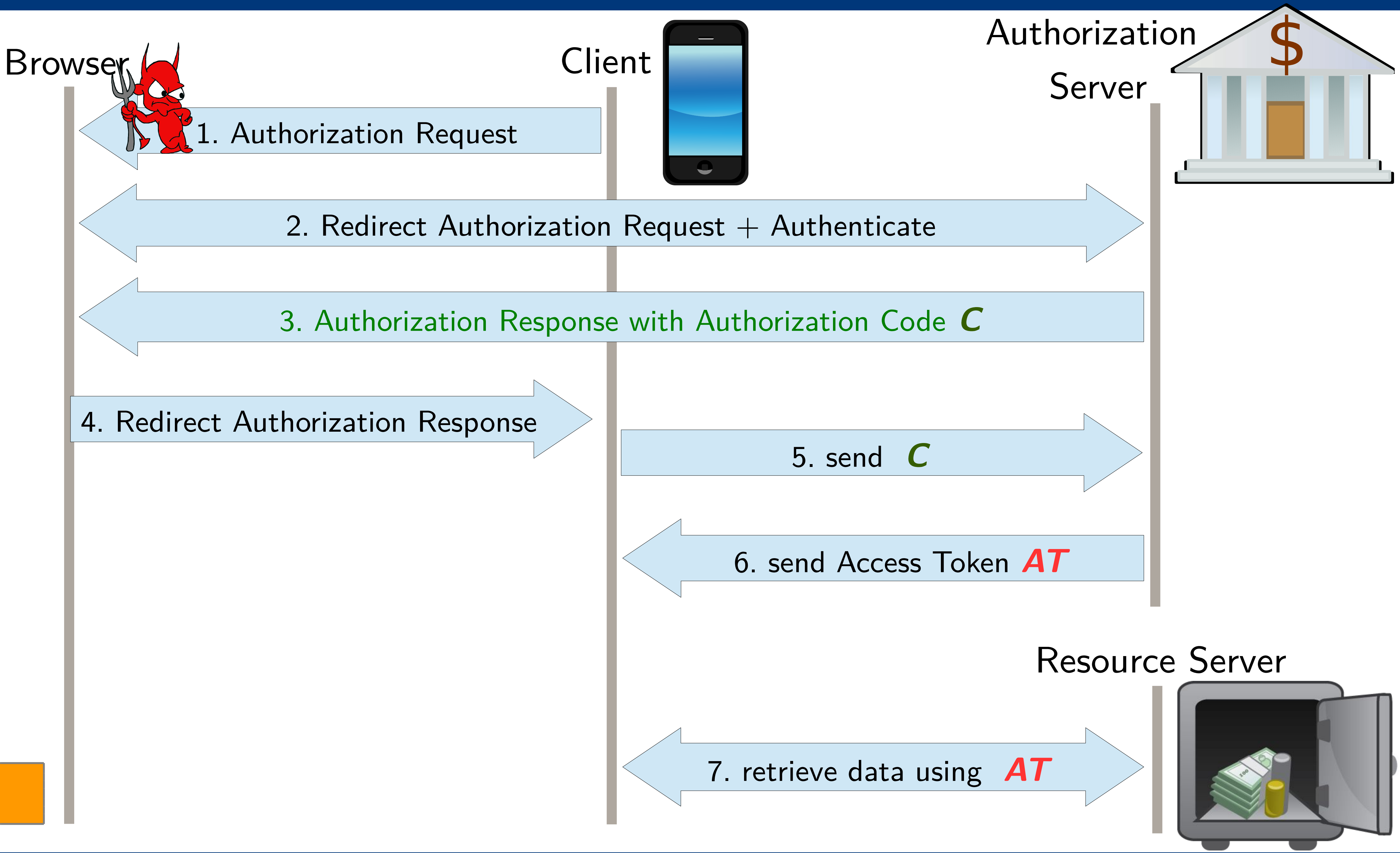

Browser
Client
Authorization
Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

FAPI: Secure, even if ...


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code C
4. Redirect Authorization Response
5. send C
6. send Access Token AT
Resource Server
7. retrieve data using AT

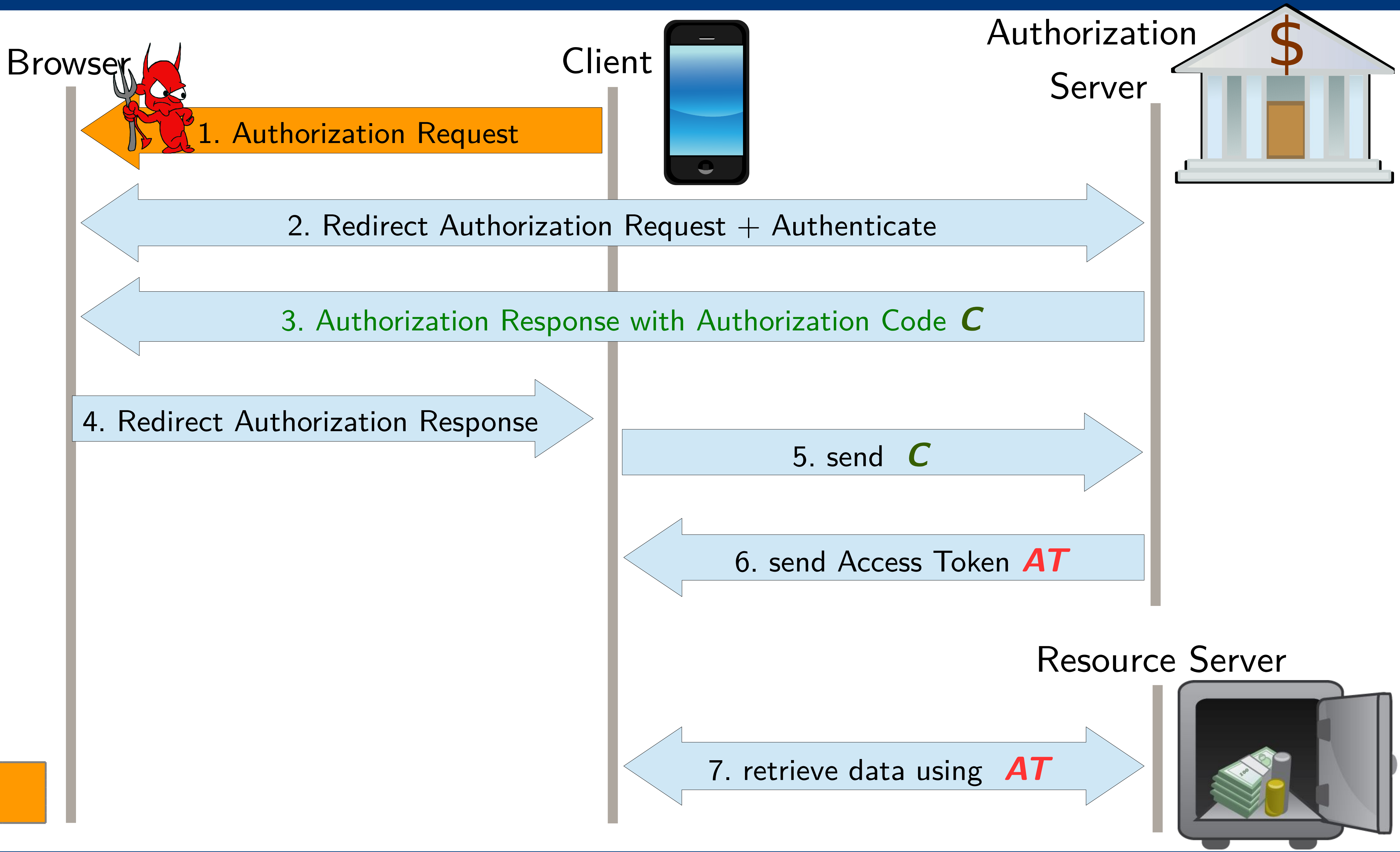
Leakage

FAPI: Secure, even if ...


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

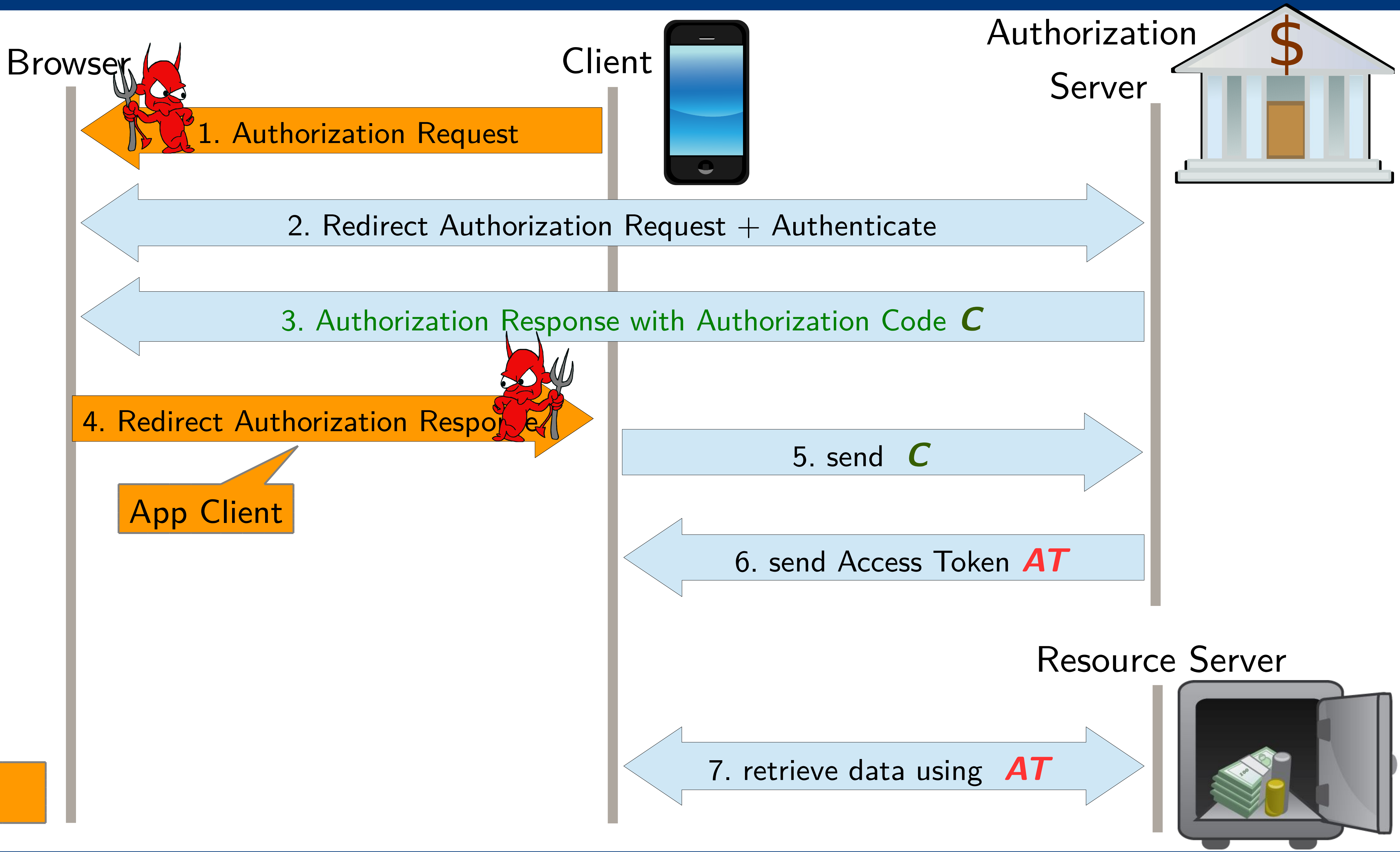
Leakage

FAPI: Secure, even if ...


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

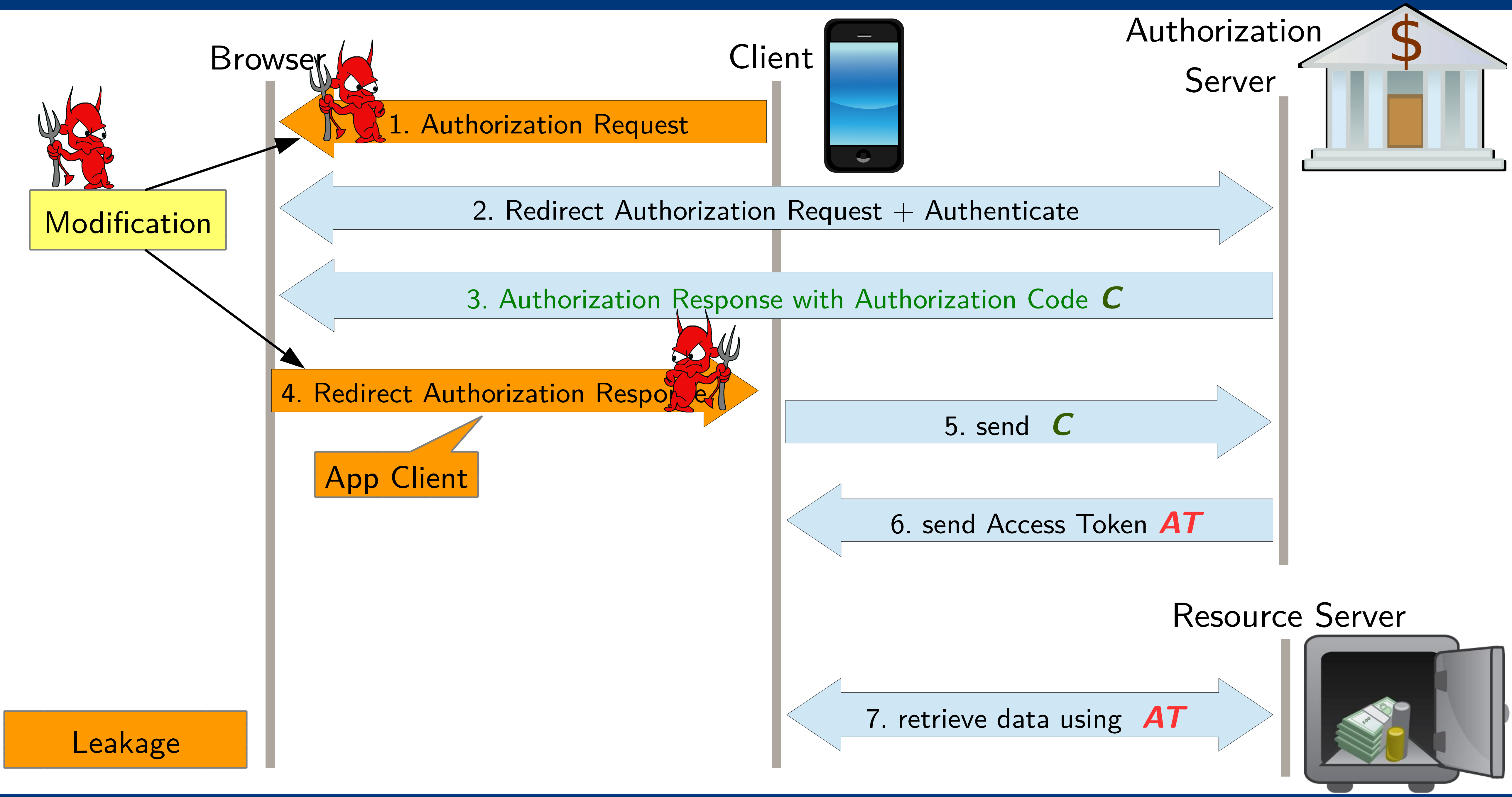
Leakage

FAPI: Secure, even if ...

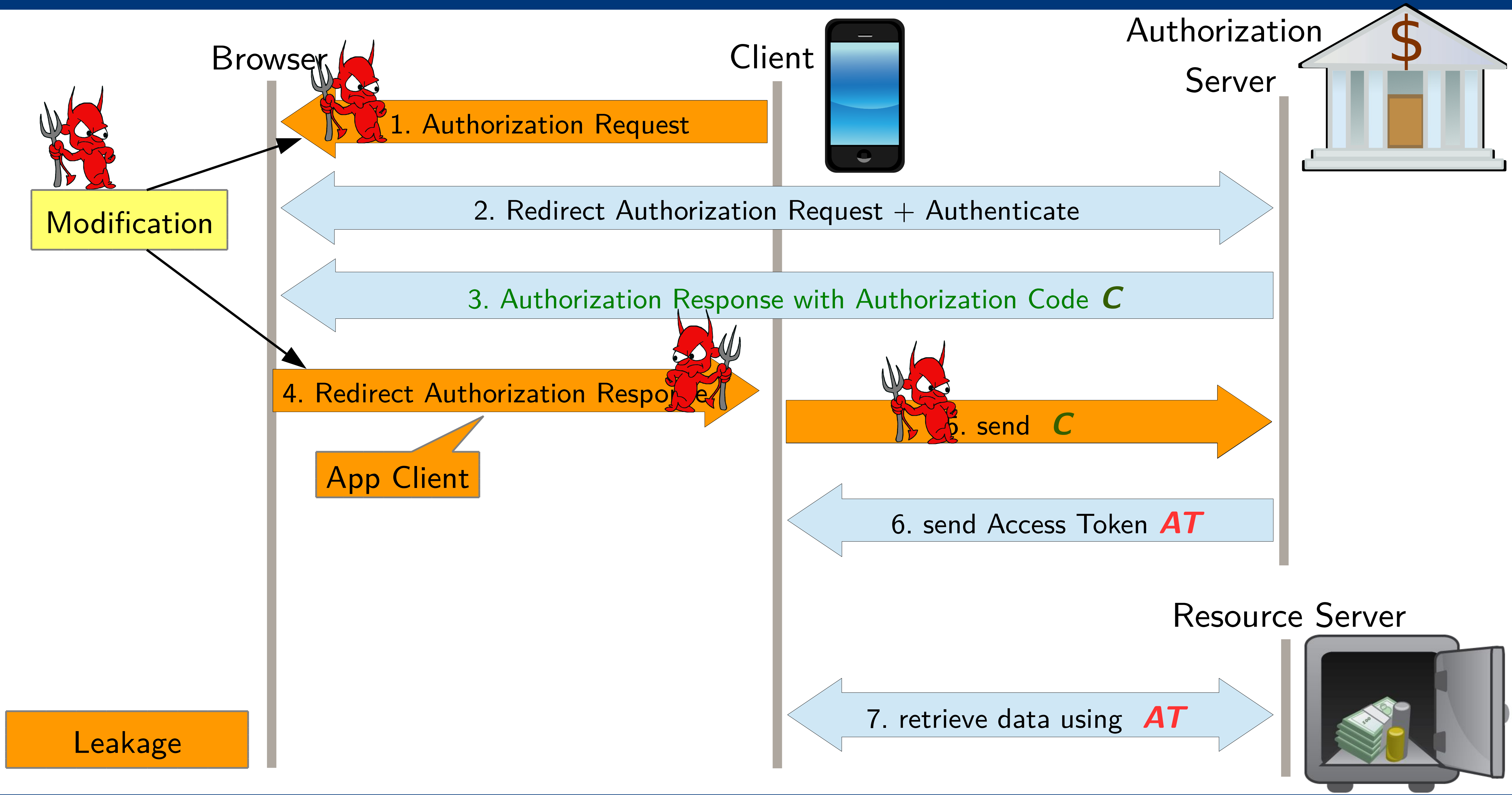

Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
App Client
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

Leakage

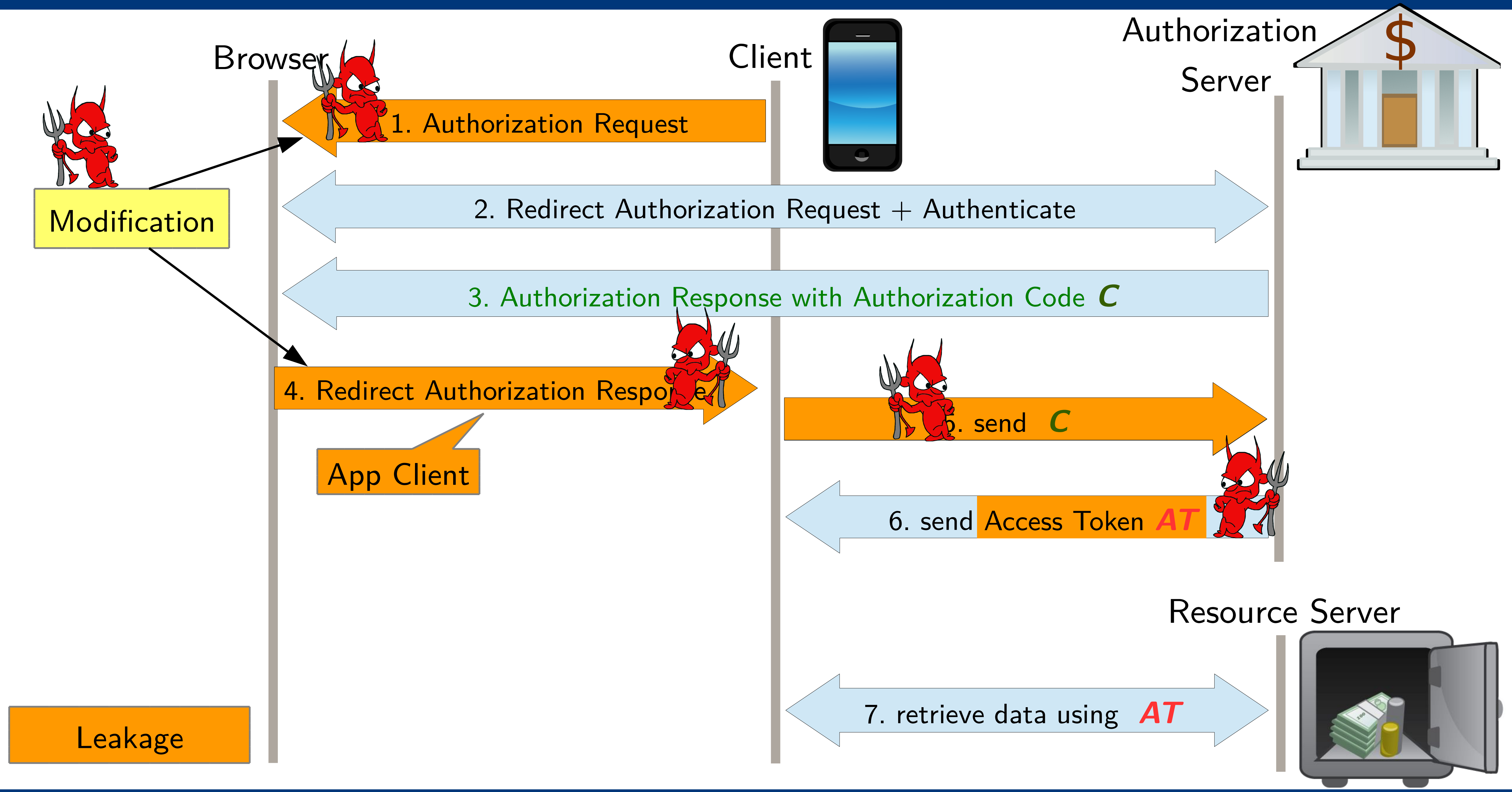
FAPI: Secure, even if ...


Browser
Client
Authorization Server
$
Modification
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code C
4. Redirect Authorization Response
App Client
5. send C
6. send Access Token AT
Resource Server
7. retrieve data using AT
Leakage

FAPI: Secure, even if ...

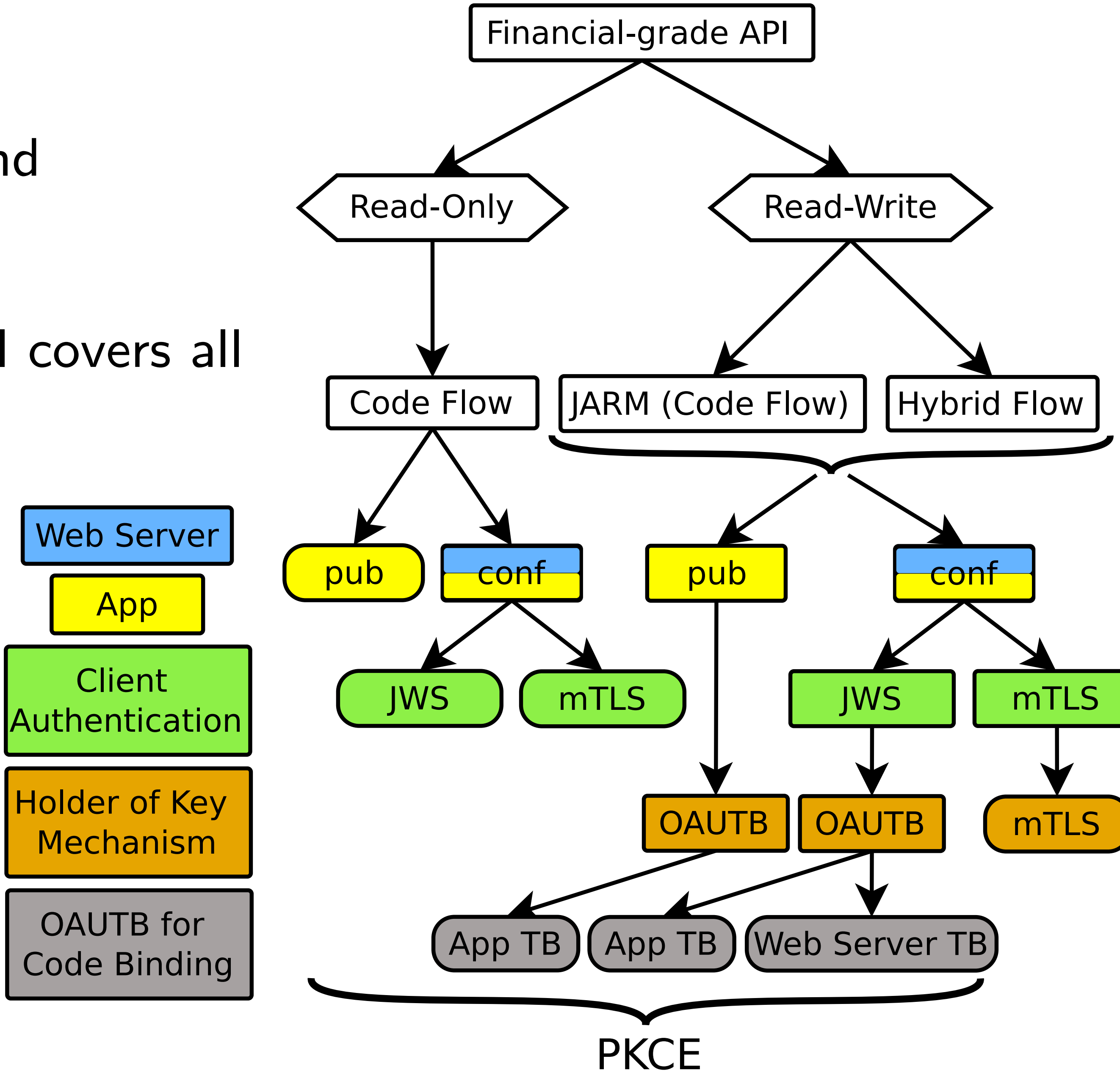

Browser
Client
Authorization Server
Modification
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
App Client
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*
Leakage

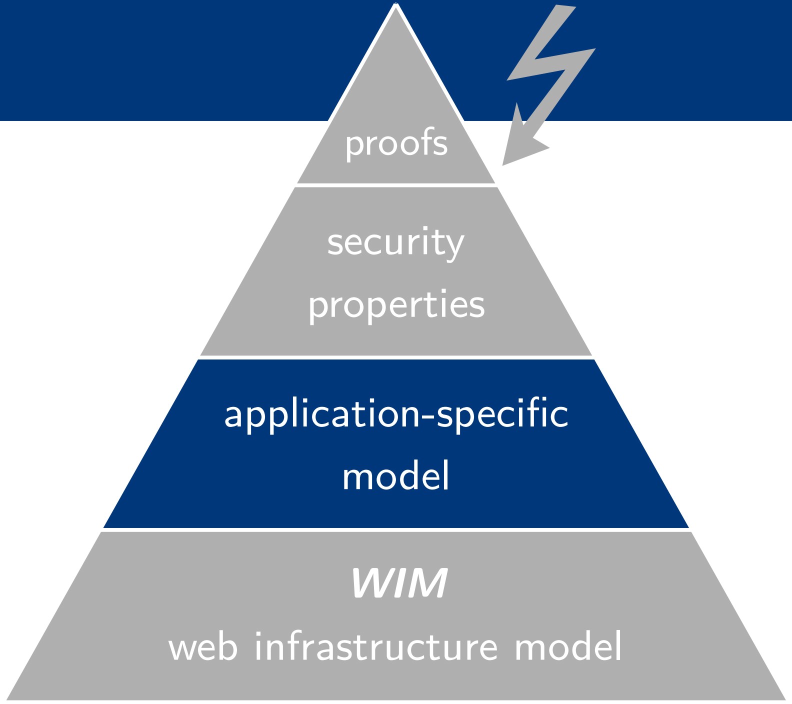
FAPI: Secure, even if ...


Browser
Client
Authorization Server
Modification
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
App Client
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*
Leakage

FAPI: Model

- FAPIa has many options and configurations
- Our WIM model covers all of them





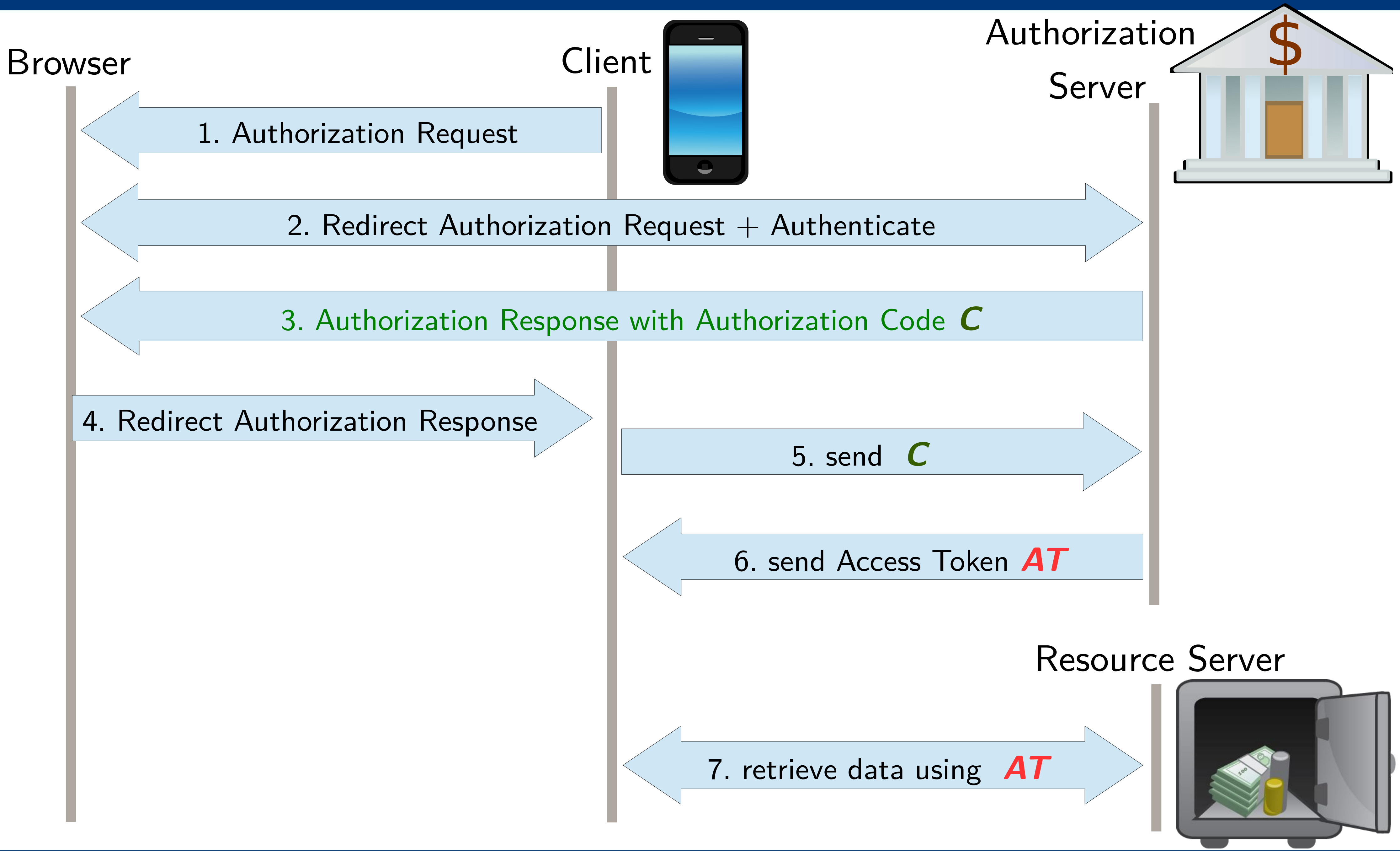
New Defense Mechanisms

- Token Binding
- Proof Key for Code Exchange (PKCE)
- Improved Client Authentication
- Signed Authorization Request
- Signed Authorization Response (JARM)

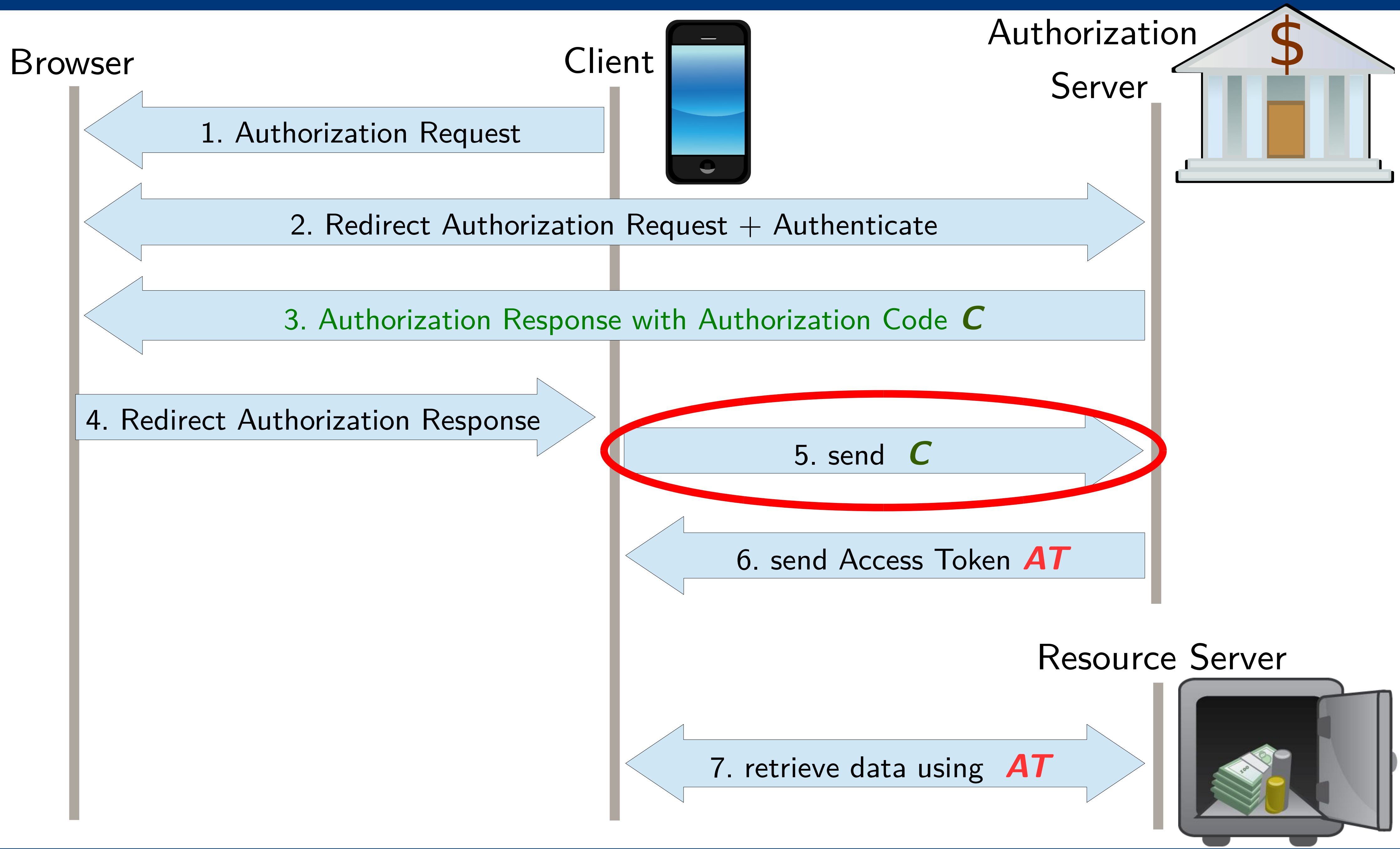
New Defense Mechanisms

- Token Binding
- Proof Key for Code Exchange (PKCE)
- Improved Client Authentication
- Signed Authorization Request
- Signed Authorization Response (JARM)

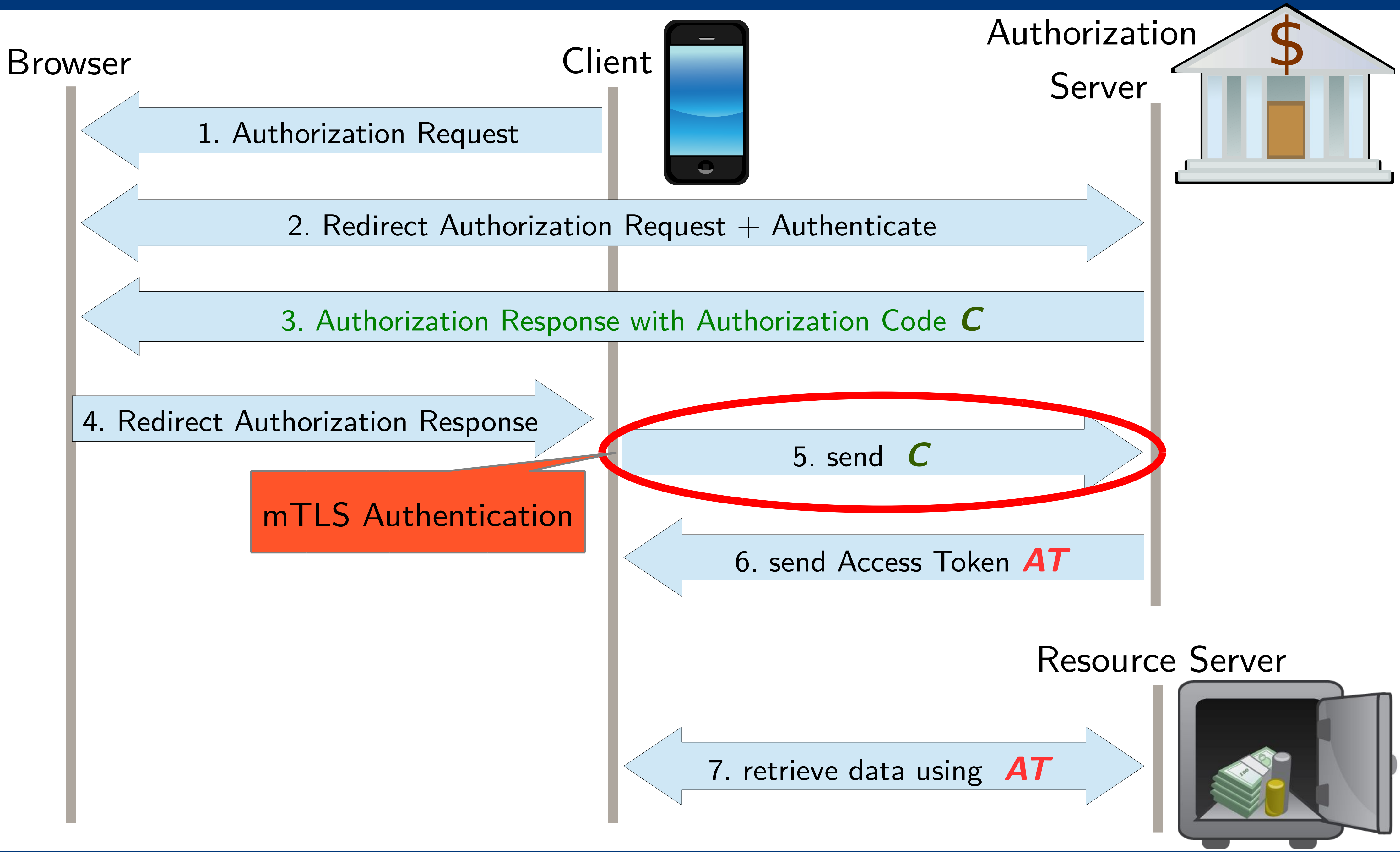
Example: Binding Access Tokens


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

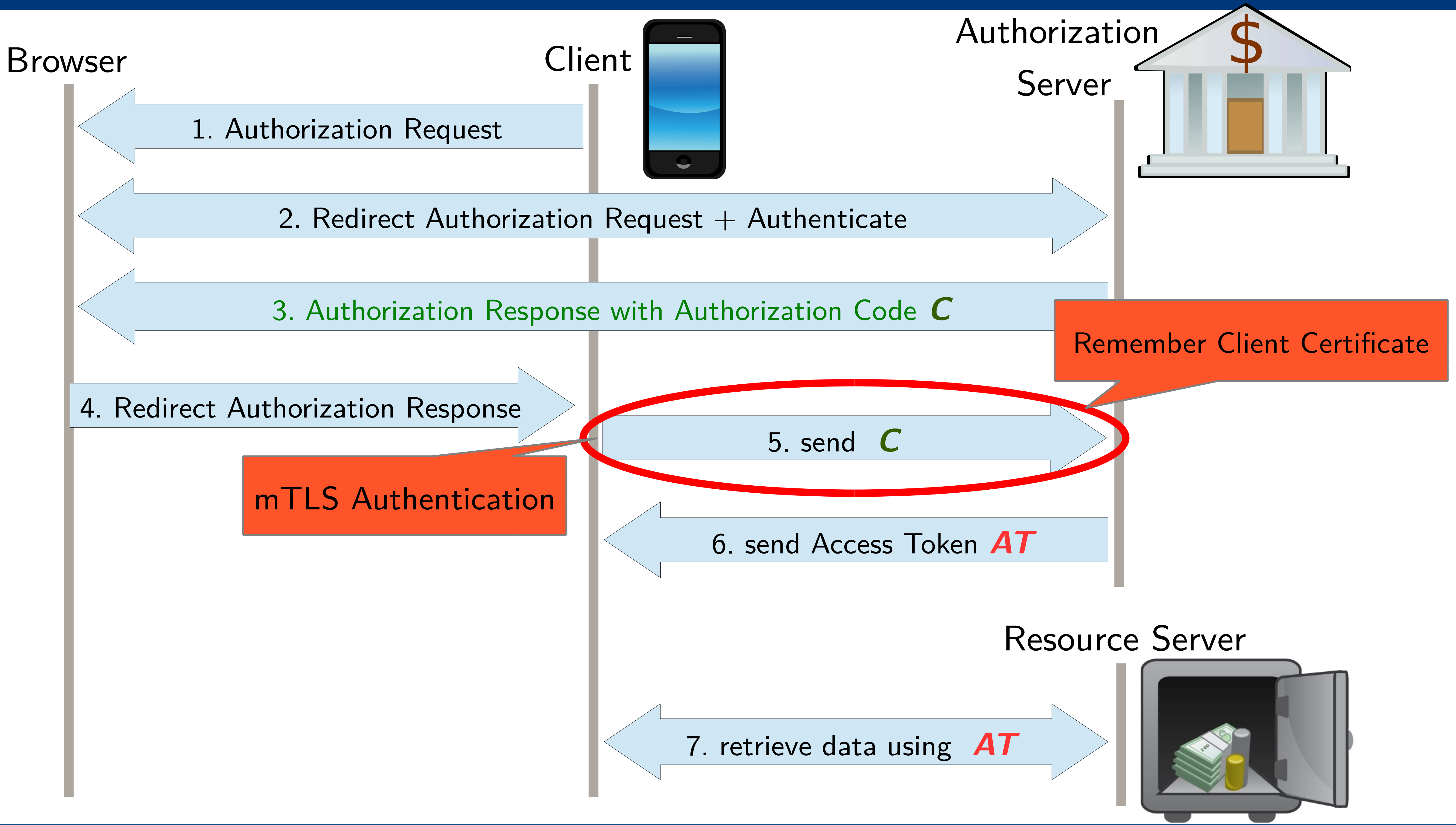
Example: Binding Access Tokens


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

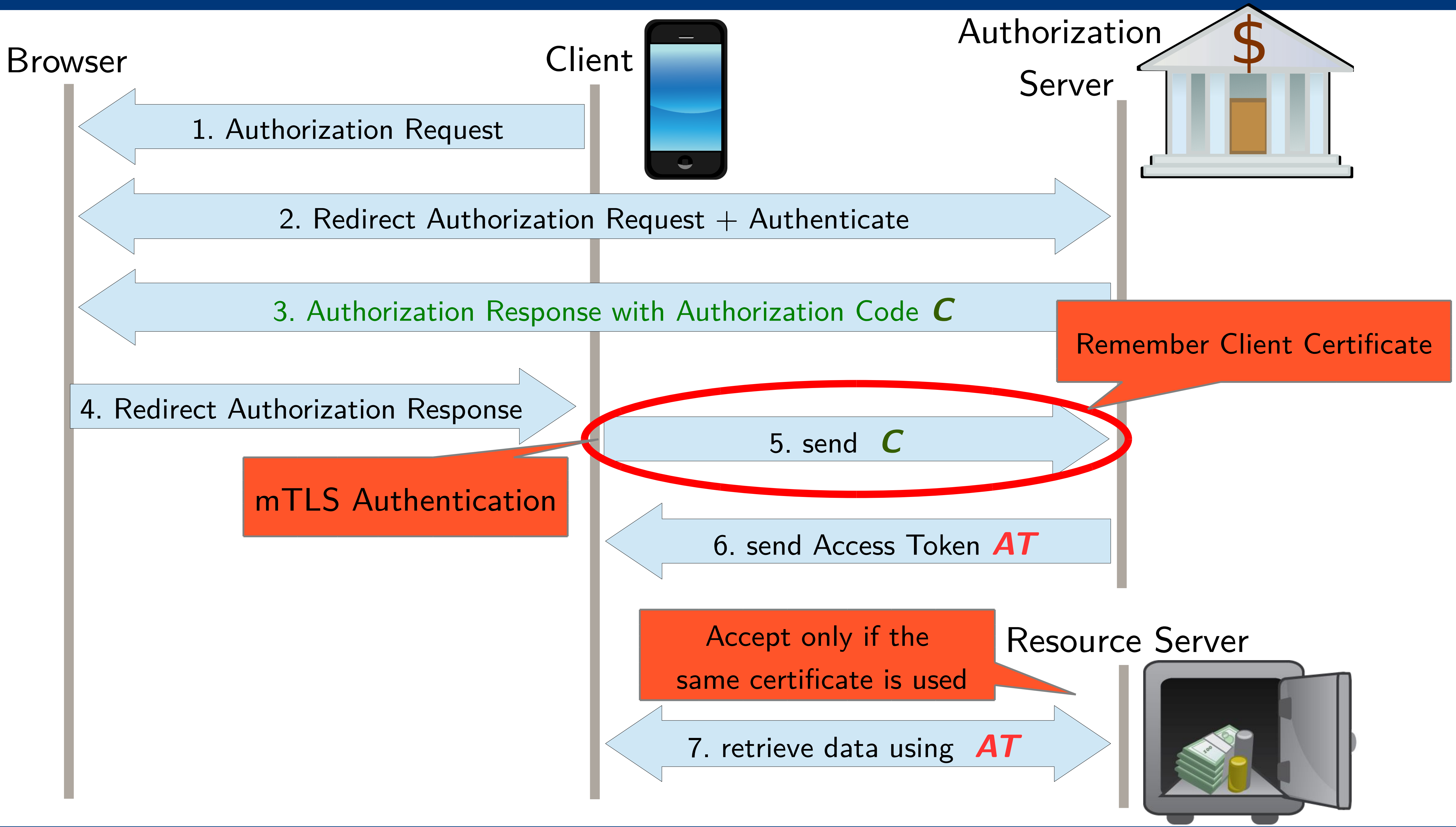
Example: Binding Access Tokens


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
mTLS Authentication
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

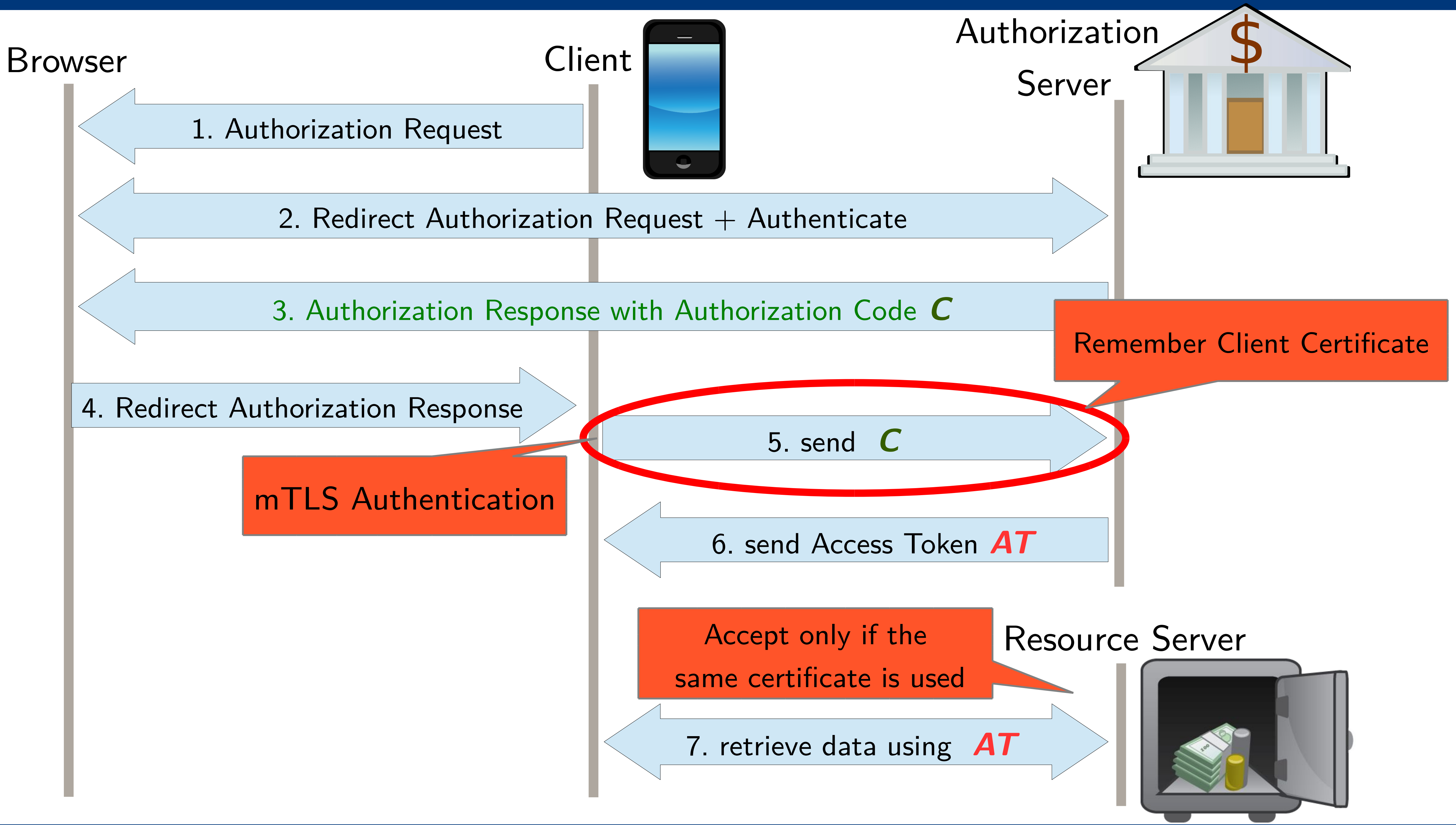
Example: Binding Access Tokens


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
Remember Client Certificate
4. Redirect Authorization Response
5. send *C*
mTLS Authentication
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

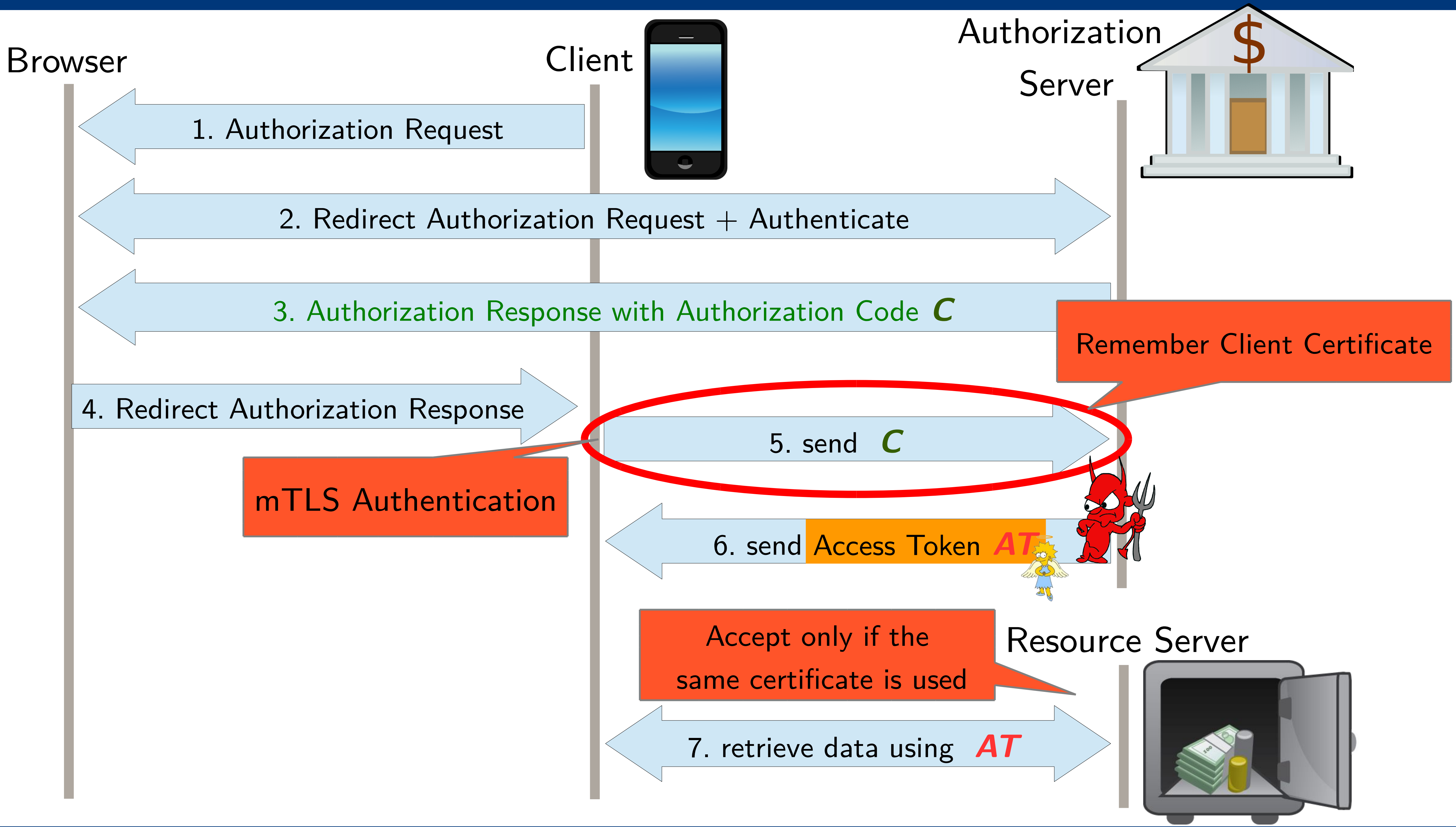
Example: Binding Access Tokens


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
Remember Client Certificate
4. Redirect Authorization Response
5. send *C*
mTLS Authentication
6. send Access Token *AT*
Accept only if the same certificate is used
Resource Server
7. retrieve data using *AT*

Example: Binding Access Tokens


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
Remember Client Certificate
4. Redirect Authorization Response
5. send *C*
mTLS Authentication
6. send Access Token *AT*
Accept only if the same certificate is used
Resource Server
7. retrieve data using *AT*

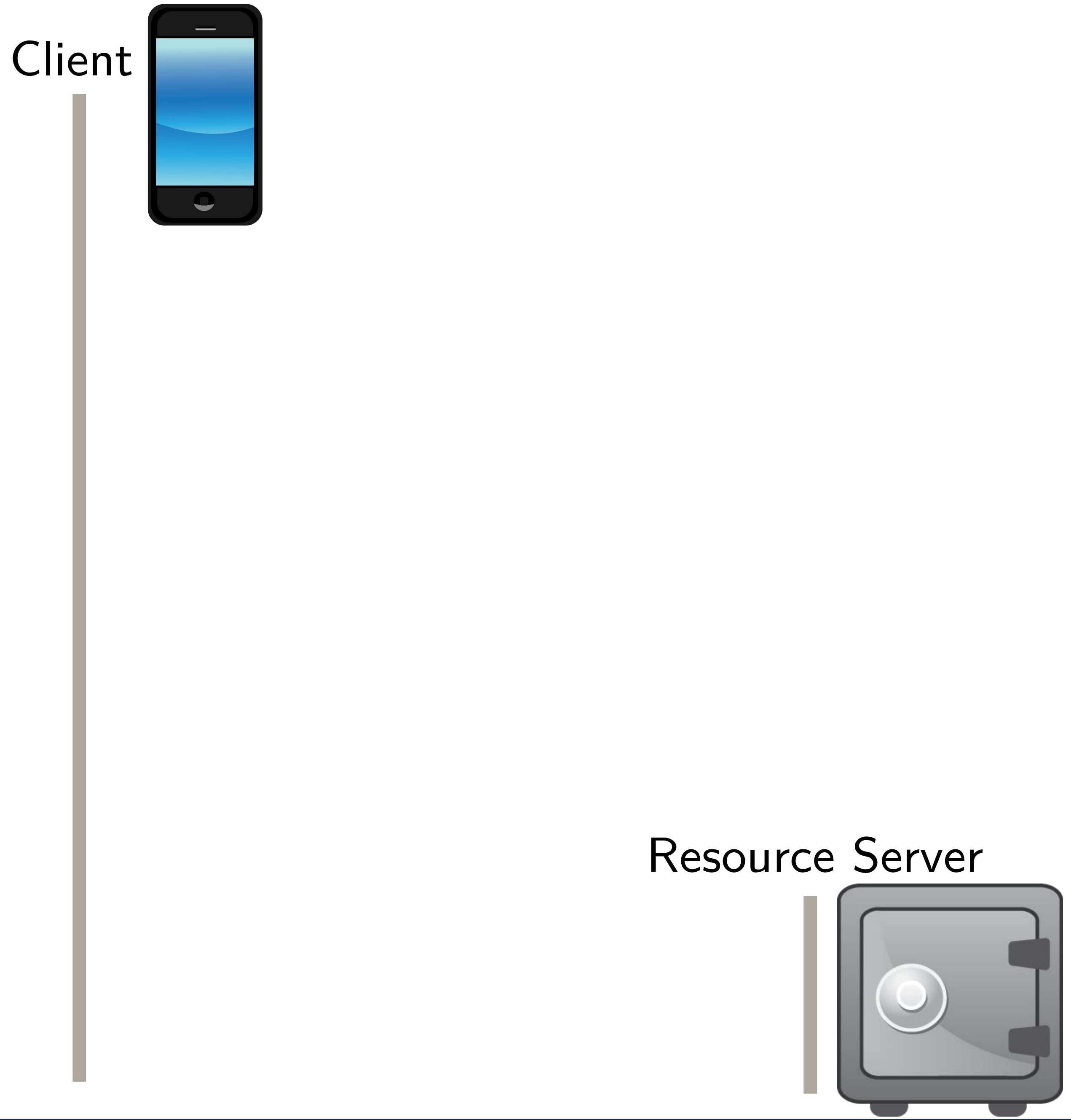
Example: Binding Access Tokens



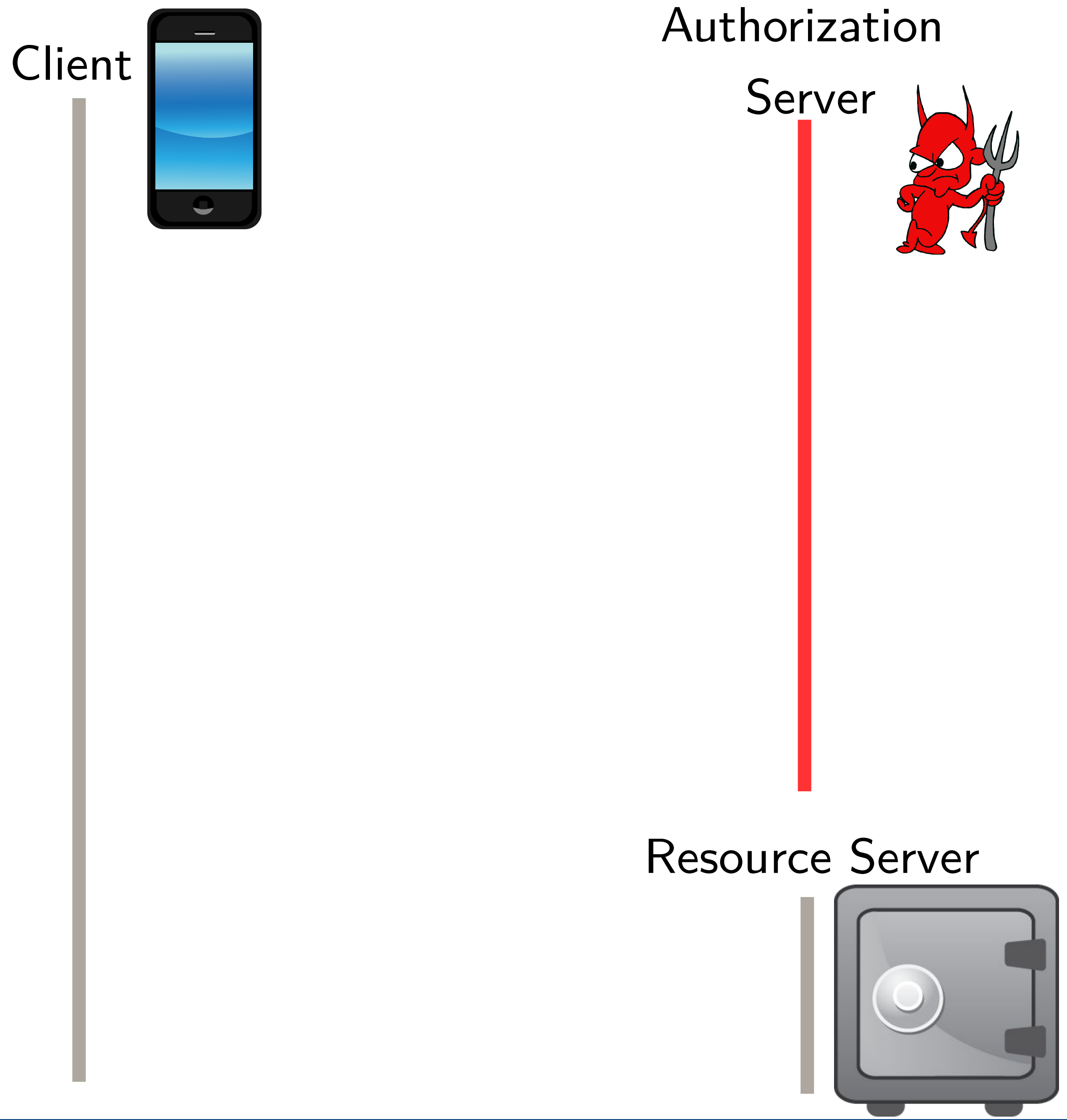
Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
Remember Client Certificate
4. Redirect Authorization Response
5. send *C*
mTLS Authentication
6. send Access Token *AT*
Accept only if the same certificate is used
Resource Server
7. retrieve data using *AT*

Cuckoo’s Token Attack

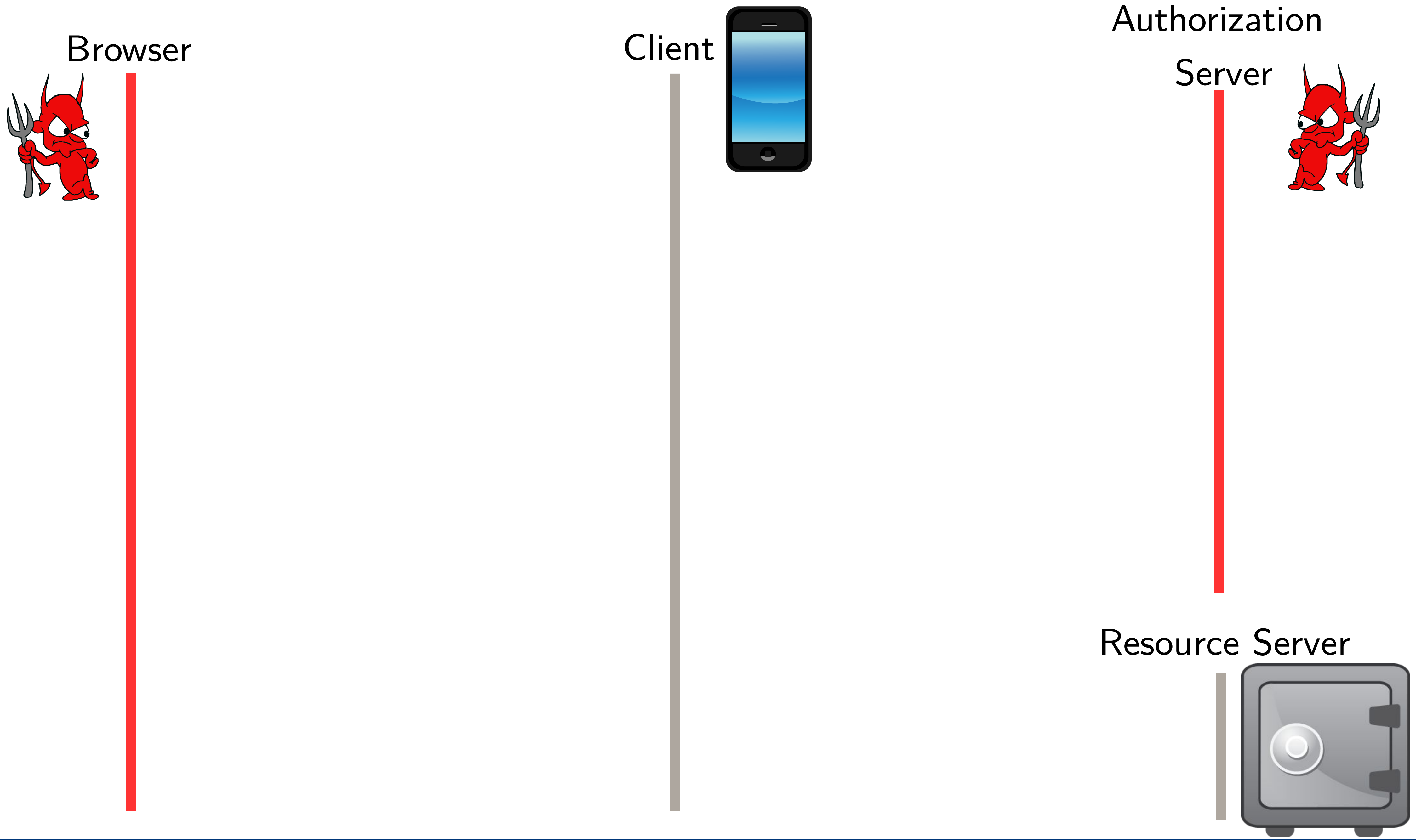
Cuckoo's Token Attack



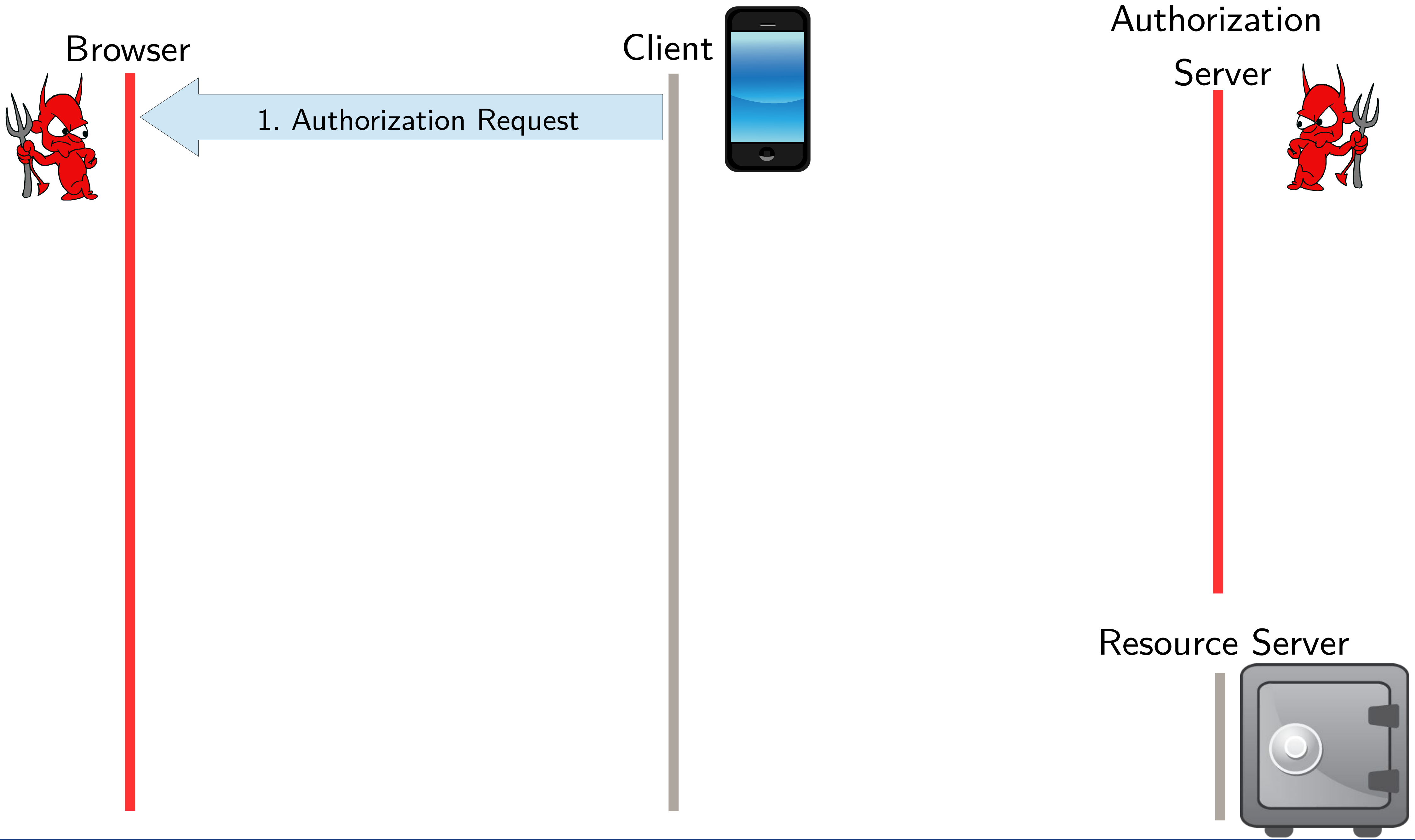
Cuckoo’s Token Attack



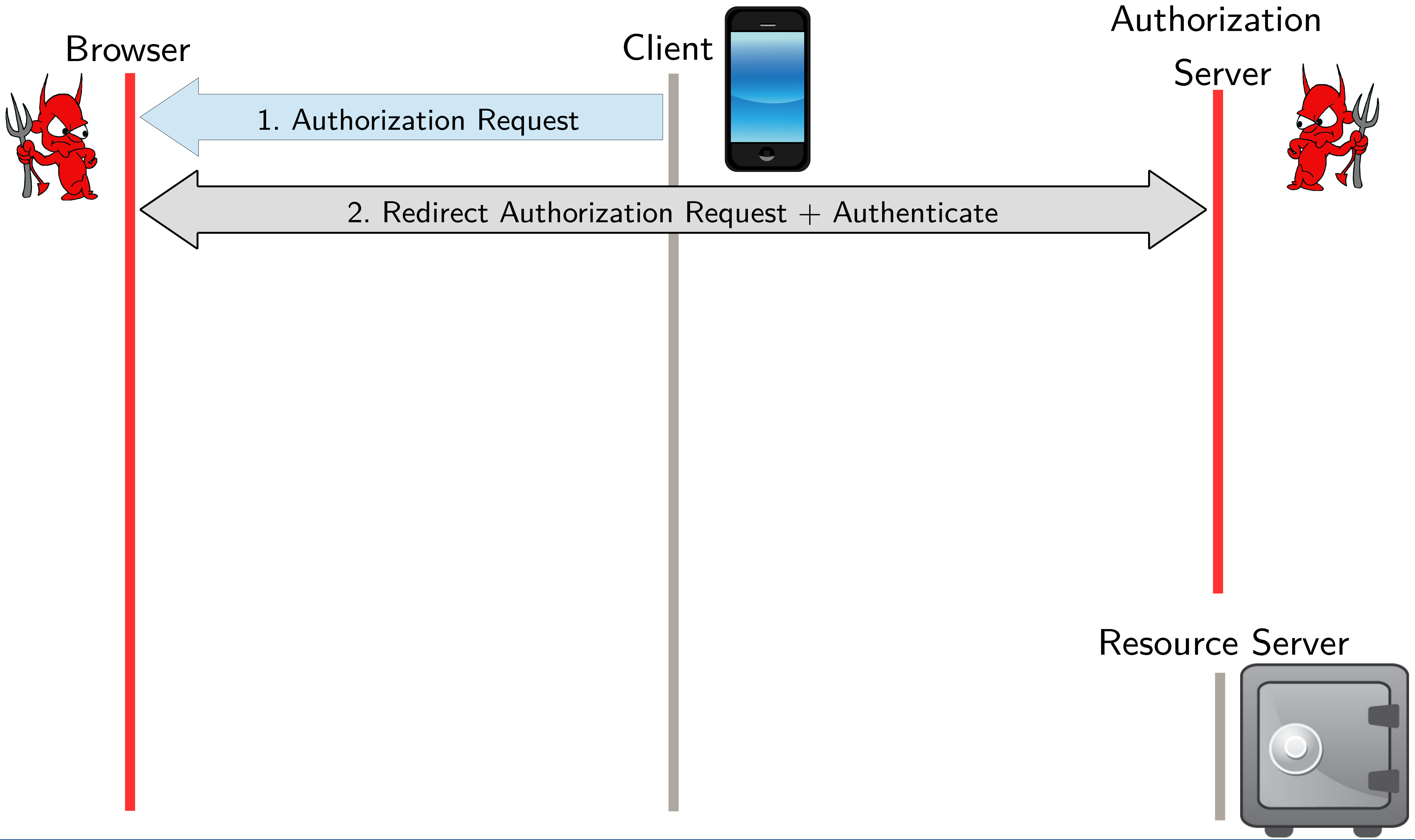
Cuckoo's Token Attack



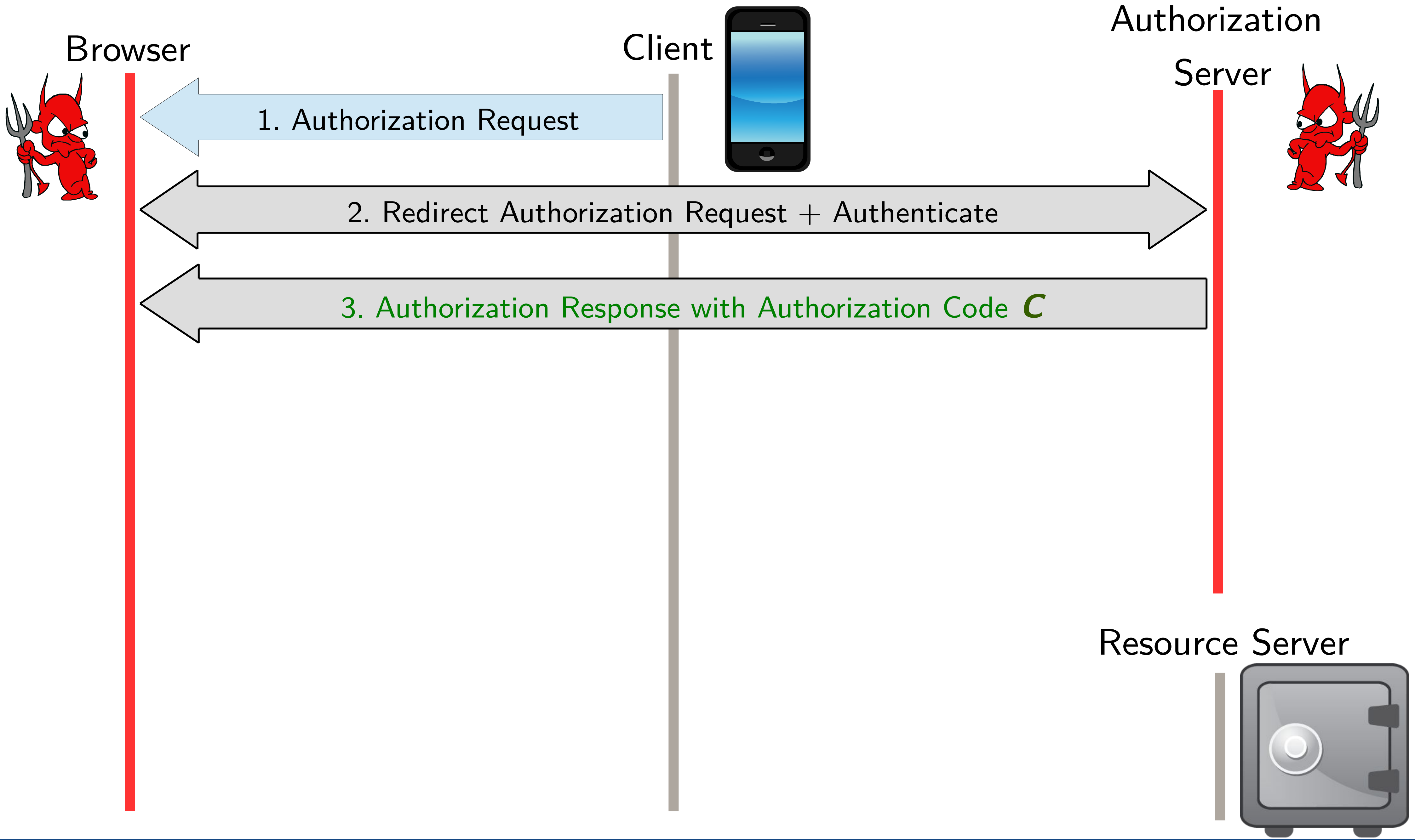
Cuckoo's Token Attack



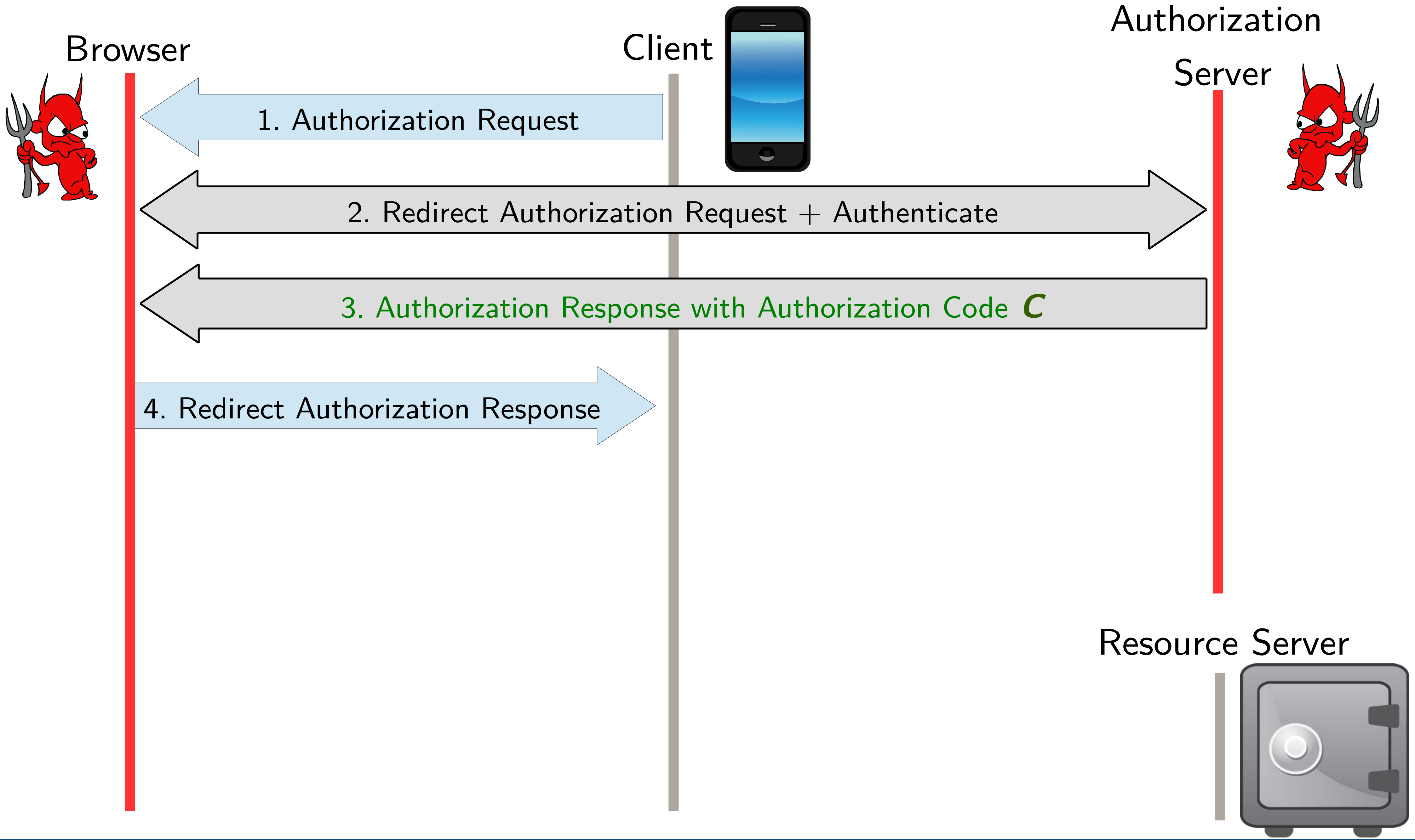
Cuckoo's Token Attack



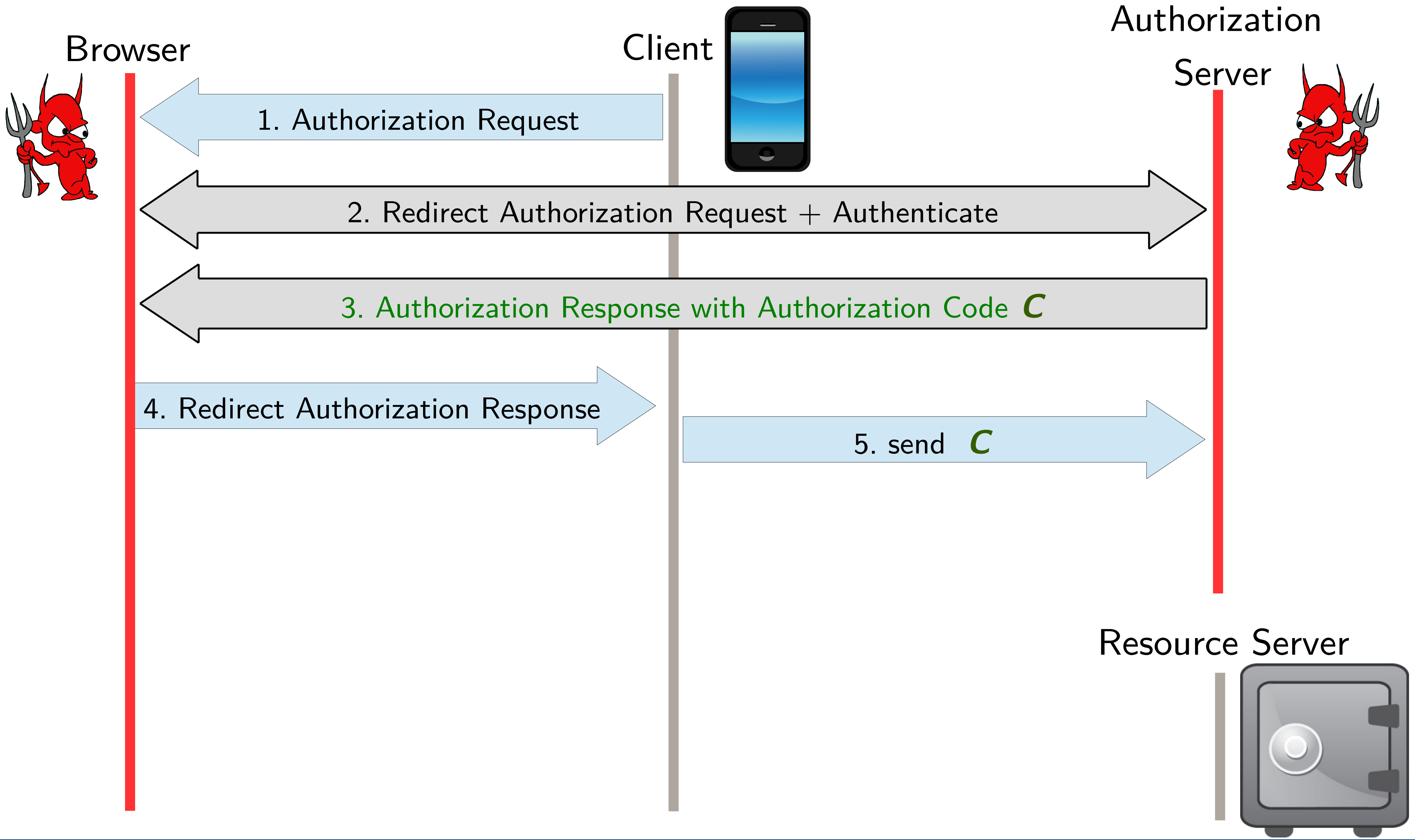
Cuckoo's Token Attack


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
Resource Server

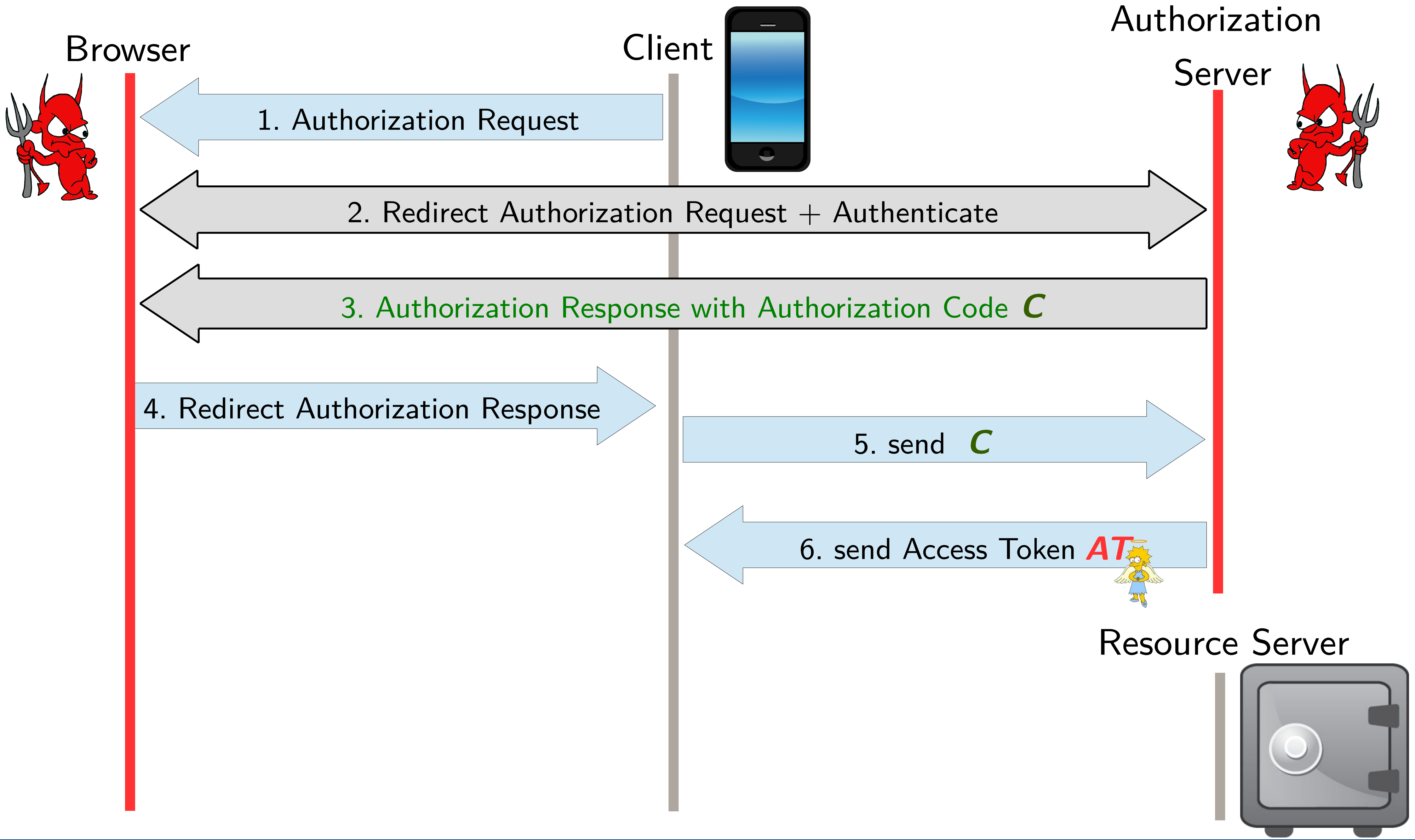
Cuckoo's Token Attack


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
Resource Server

Cuckoo's Token Attack

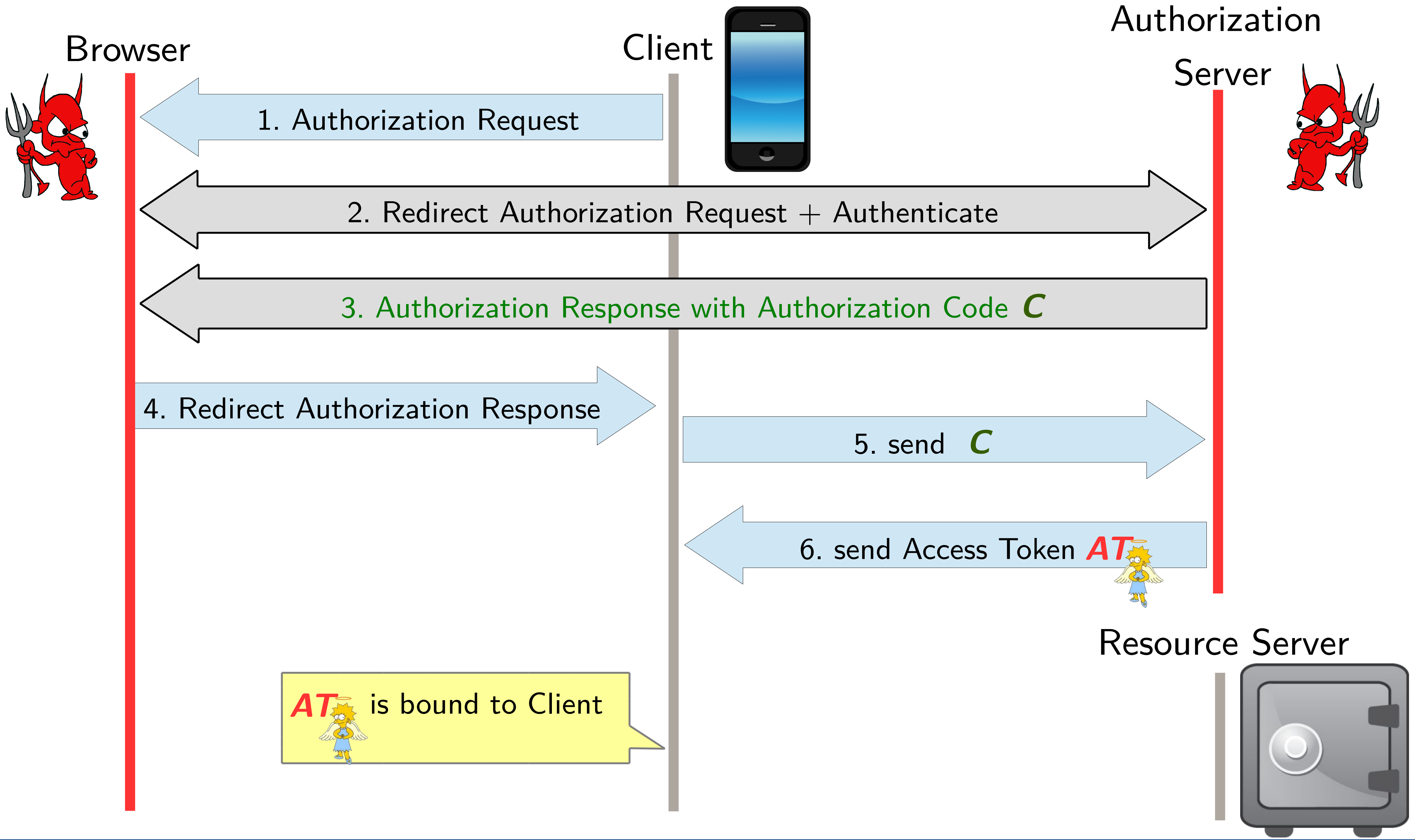

Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code C
4. Redirect Authorization Response
5. send C
Resource Server

Cuckoo's Token Attack

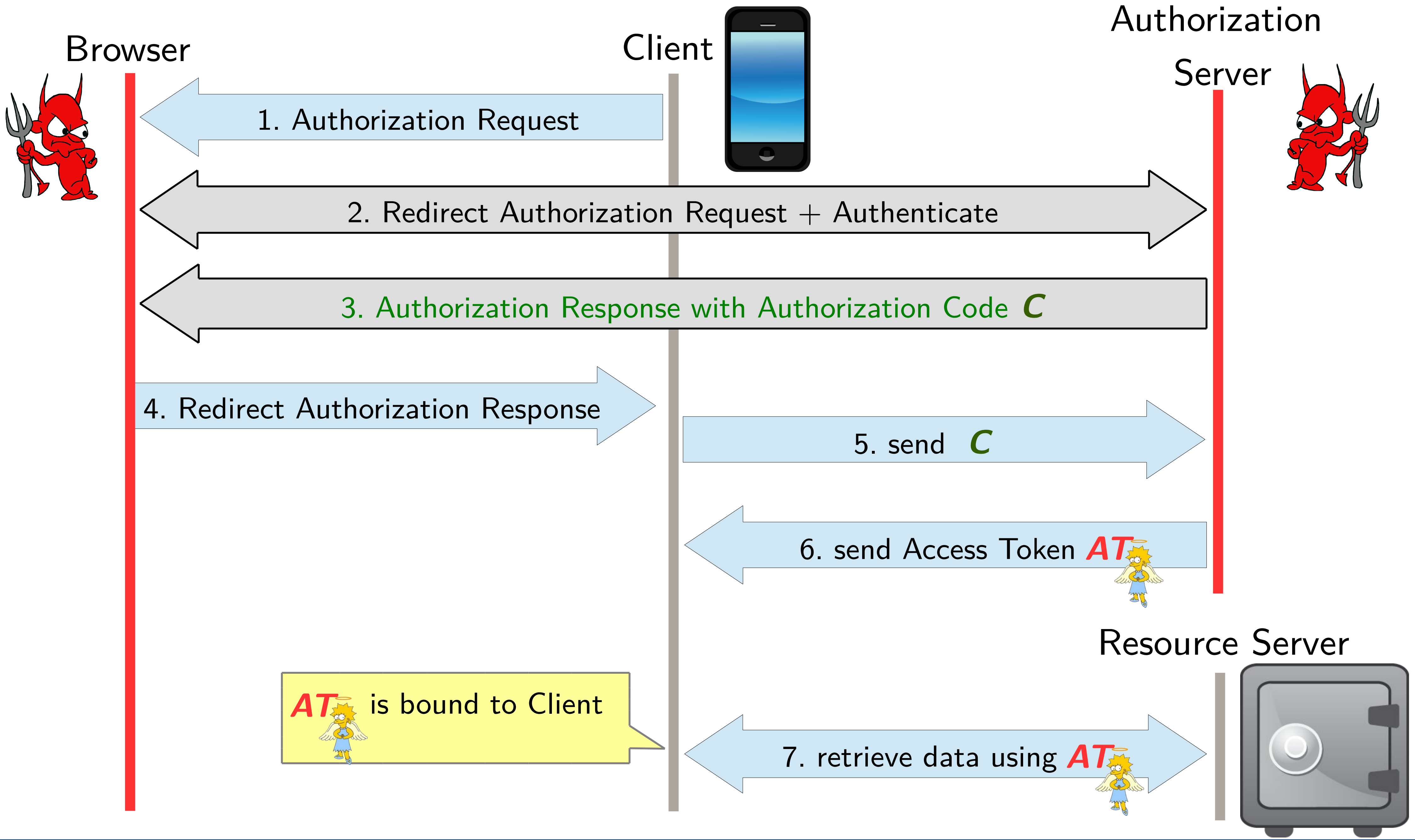


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server

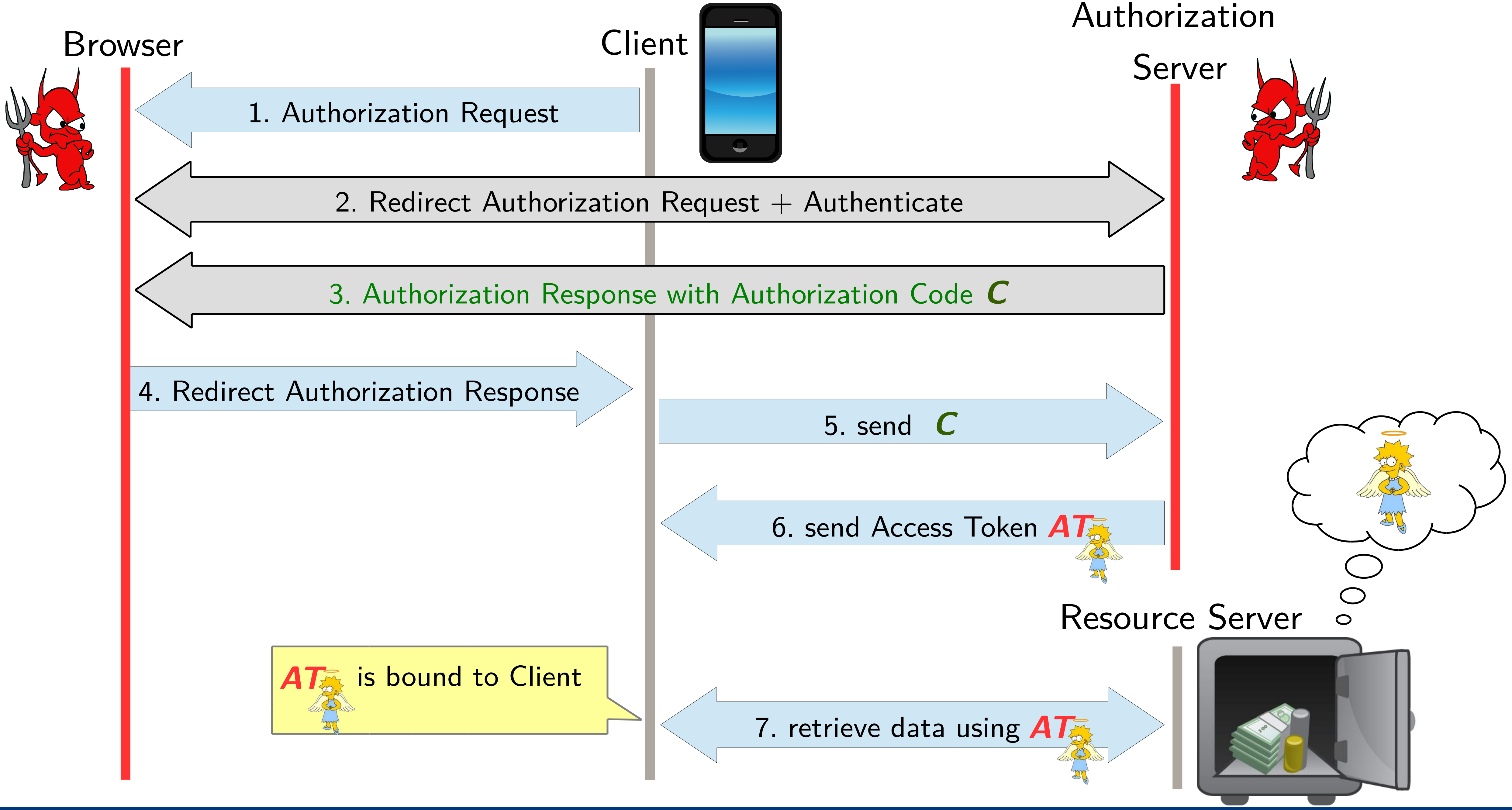
Cuckoo's Token Attack


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server
AT is bound to Client

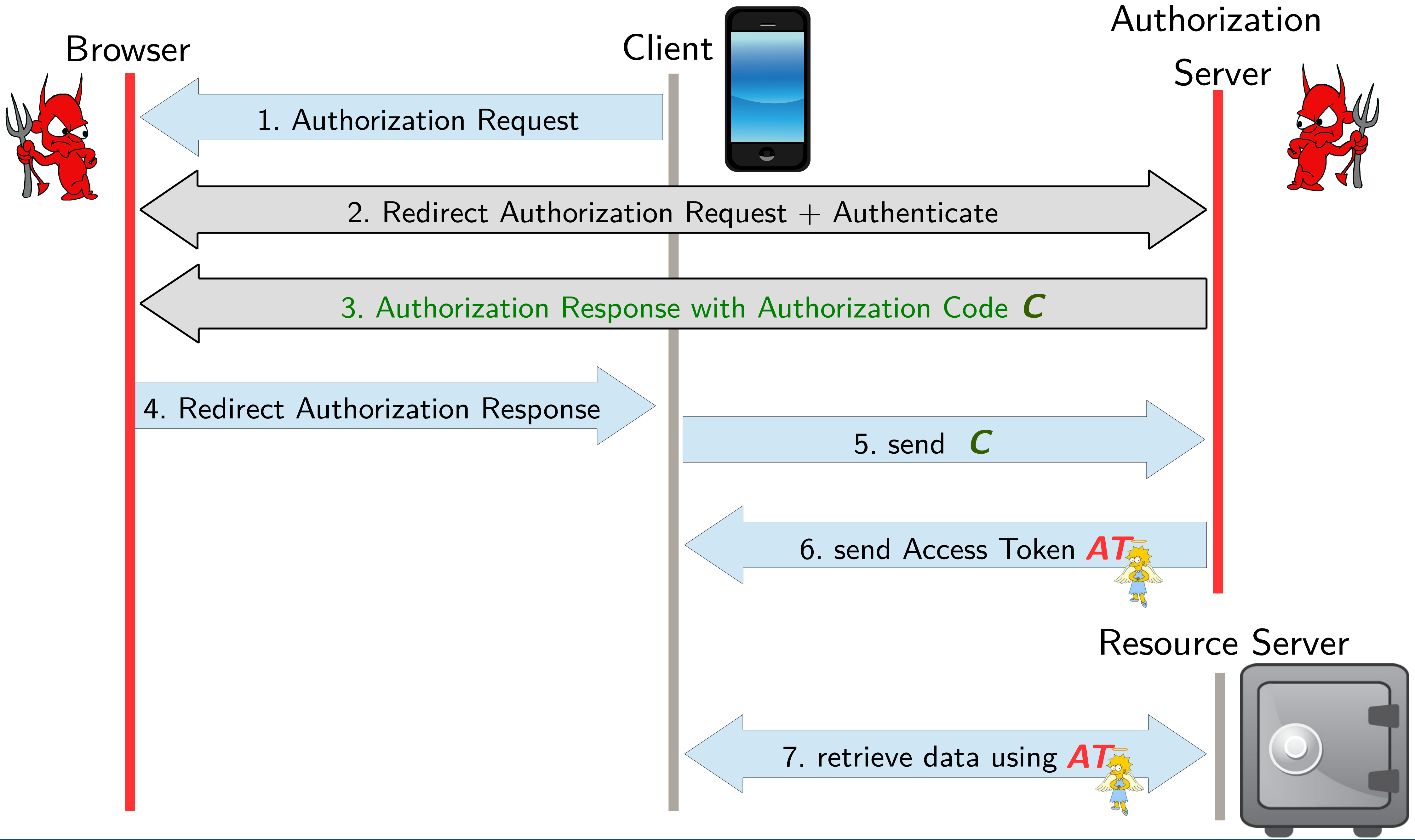
Cuckoo's Token Attack


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server
AT is bound to Client
7. retrieve data using *AT*

Cuckoo's Token Attack

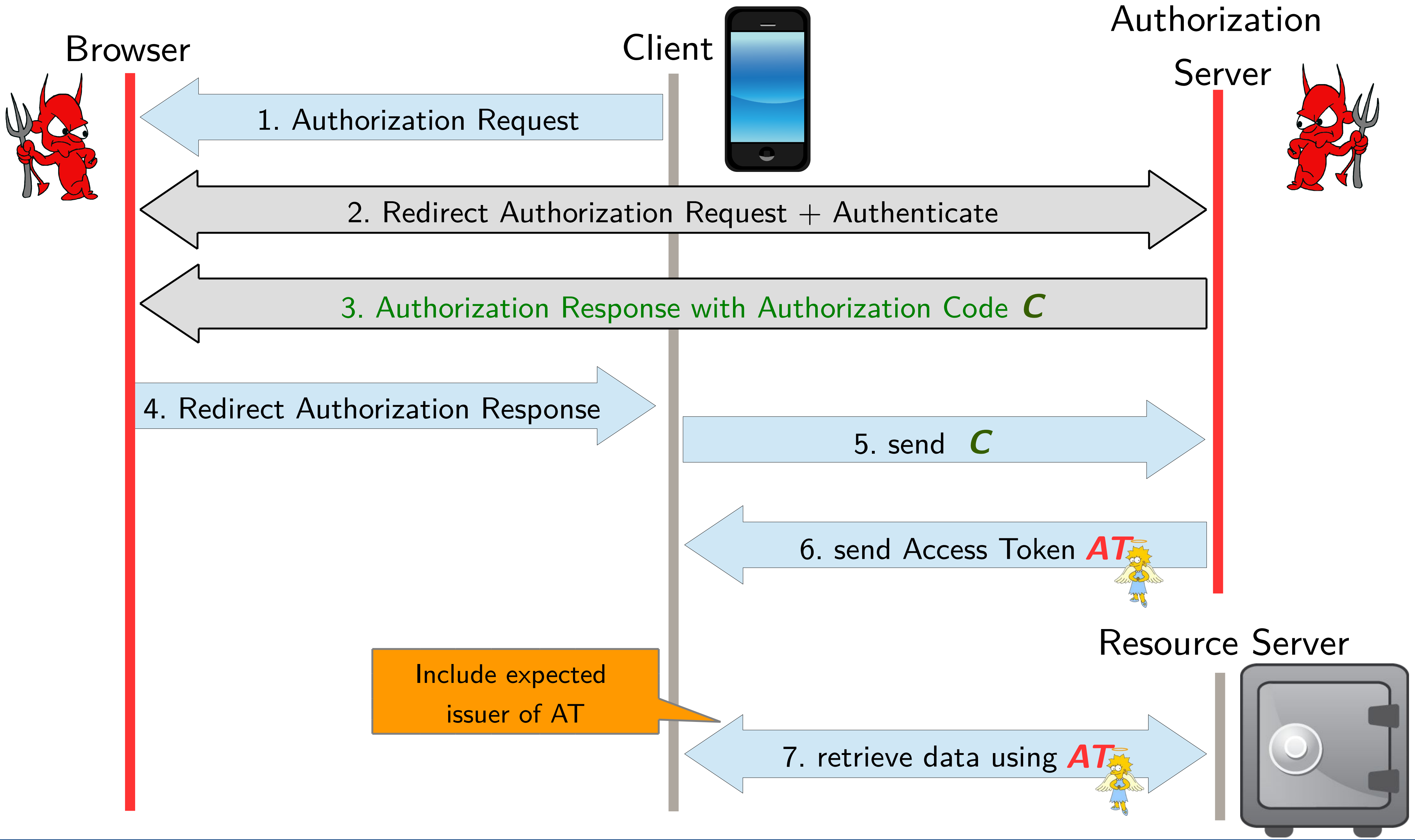

Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code C
4. Redirect Authorization Response
5. send C
6. send Access Token AT
Resource Server
AT is bound to Client
7. retrieve data using AT

Mitigation

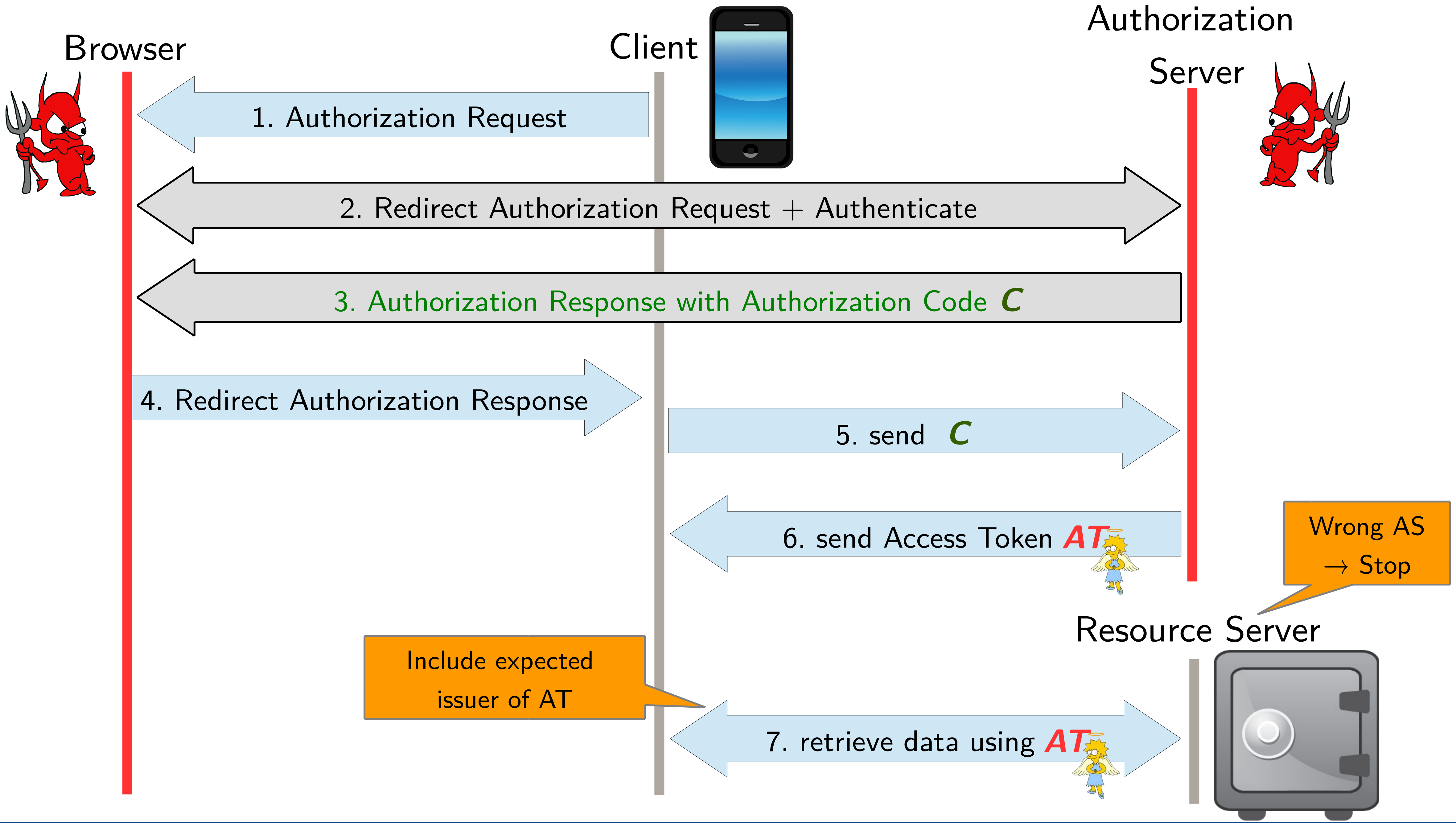


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code *C*
4. Redirect Authorization Response
5. send *C*
6. send Access Token *AT*
Resource Server
7. retrieve data using *AT*

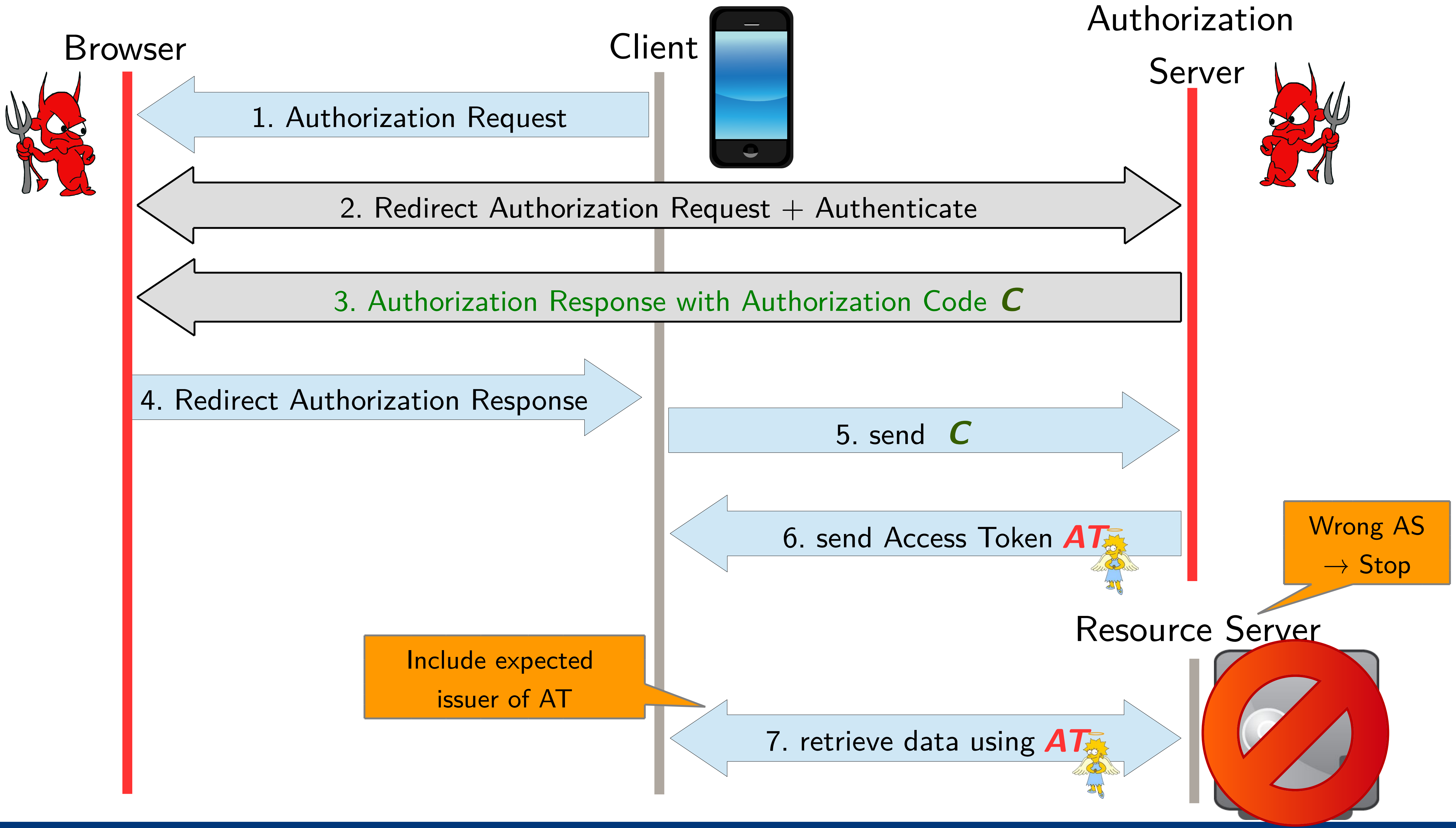
Mitigation


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code C
4. Redirect Authorization Response
5. send C
6. send Access Token AT
Resource Server
Include expected issuer of AT
7. retrieve data using AT

Mitigation

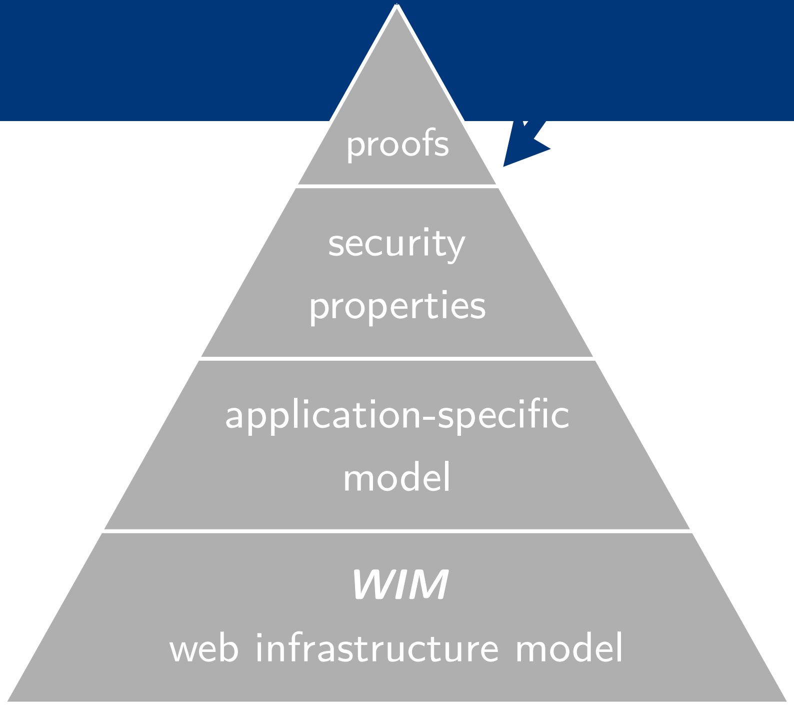

Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code C
4. Redirect Authorization Response
5. send C
6. send Access Token AT
Wrong AS → Stop
Resource Server
Include expected issuer of AT
7. retrieve data using AT

Mitigation


Browser
Client
Authorization Server
1. Authorization Request
2. Redirect Authorization Request + Authenticate
3. Authorization Response with Authorization Code C
4. Redirect Authorization Response
5. send C
6. send Access Token AT
Wrong AS → Stop
Resource Server
Include expected issuer of AT
7. retrieve data using AT

Attacks Found Through Our Formal Analysis

- Cuckoo's Token Attack
- Access Token Injection
- PKCE Chosen Challenge Attack
- Authorization Request Leak Attacks



Fixes and Security Proof

Fixes and Security Proof

- We proposed fixes for all attacks
  (again in collaboration with standardization bodies)

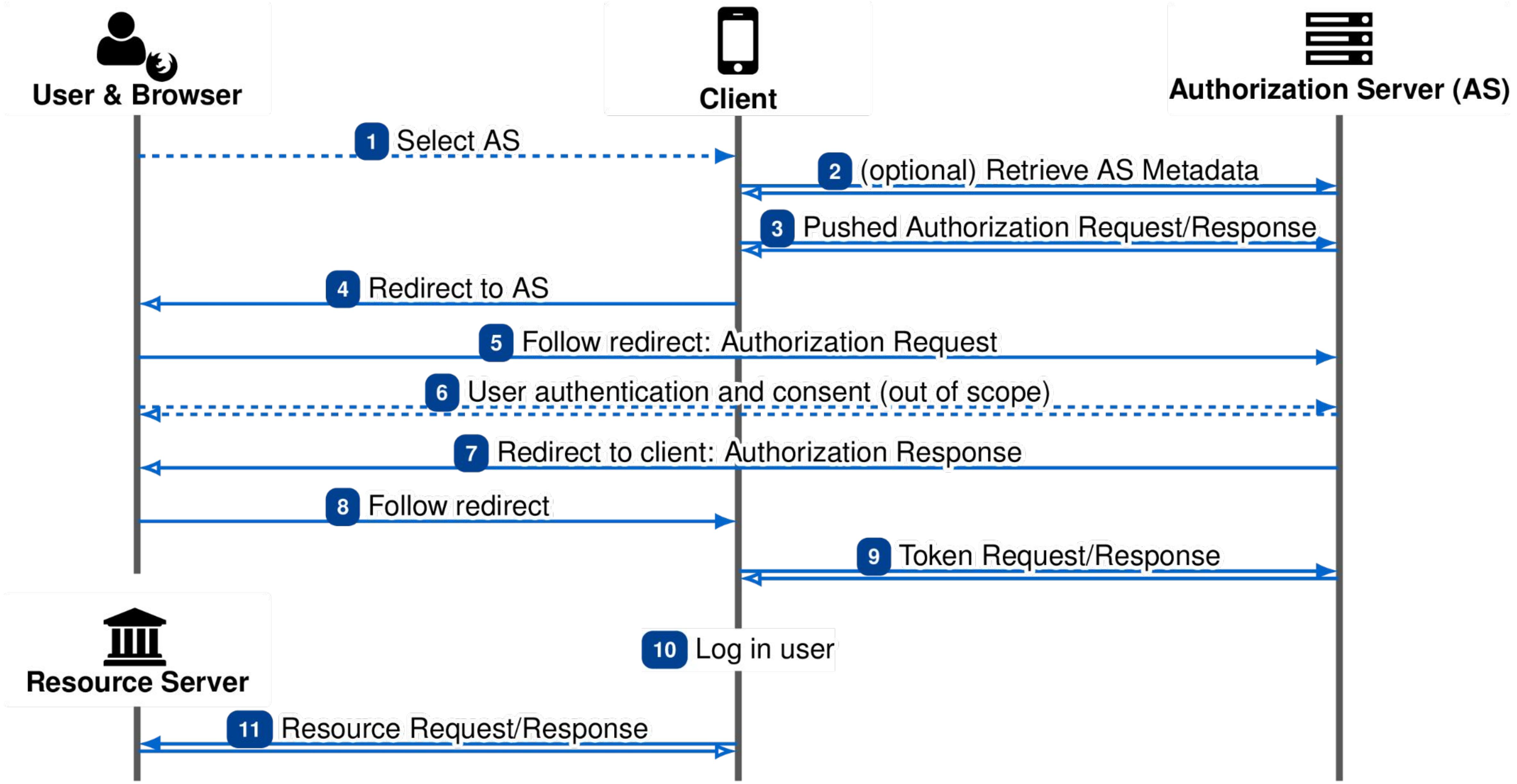
Fixes and Security Proof

- We proposed fixes for all attacks (again in collaboration with standardization bodies)
- Proved security in the WIM
  - Authentication ✓
  - Authorization ✓
  - Session Integrity ✓

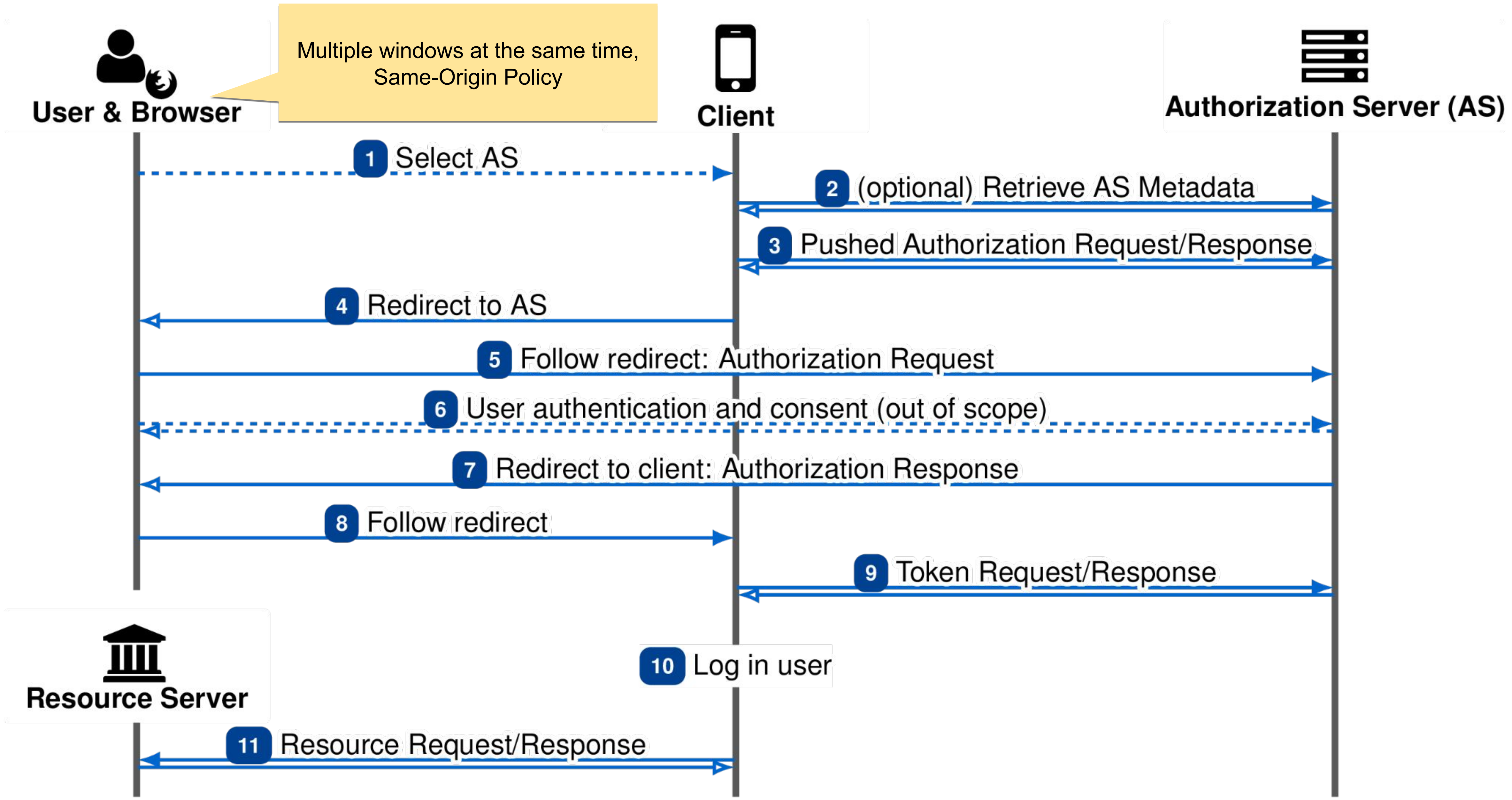
Why not just use “vanilla” DY model like for crypto protocol analysis instead of WIM?

Why not just use “vanilla” DY model instead of WIM?

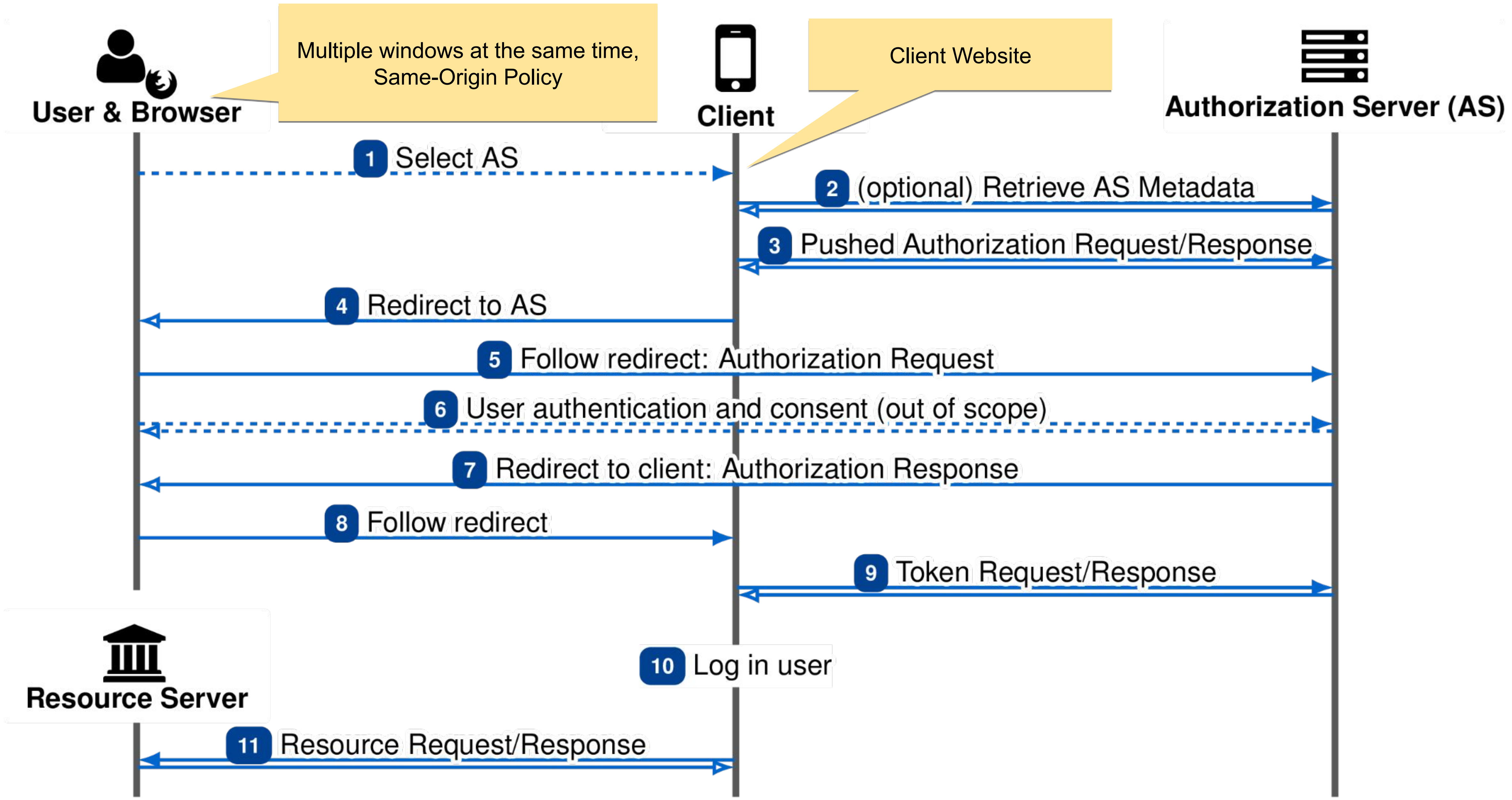
Why not just use “vanilla” DY model instead of WIM?


User & Browser
Client
Authorization Server (AS)
1 Select AS
2 (optional) Retrieve AS Metadata
3 Pushed Authorization Request/Response
4 Redirect to AS
5 Follow redirect: Authorization Request
6 User authentication and consent (out of scope)
7 Redirect to client: Authorization Response
8 Follow redirect
9 Token Request/Response
Resource Server
10 Log in user
11 Resource Request/Response

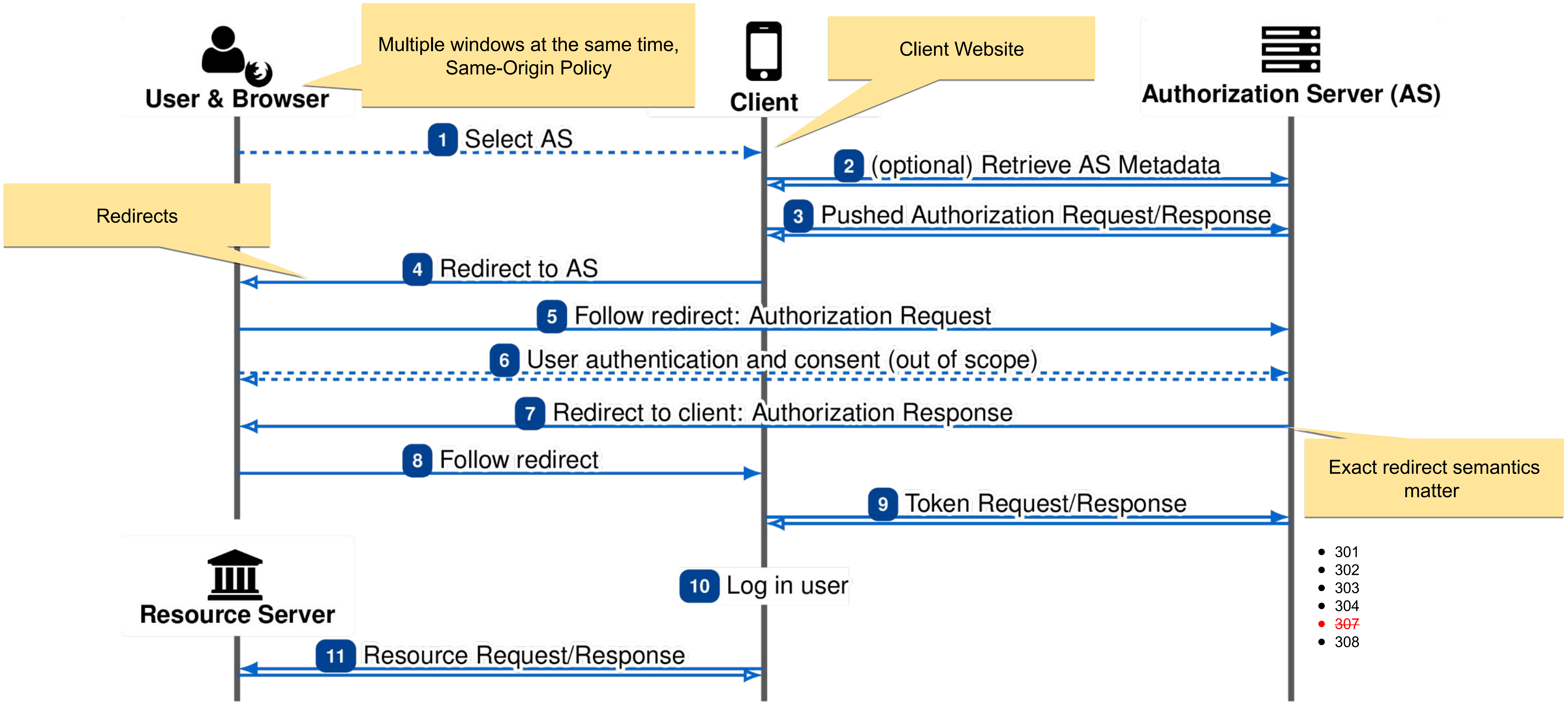
Why not just use “vanilla” DY model instead of WIM?


Multiple windows at the same time, Same-Origin Policy
User & Browser
Client
Authorization Server (AS)
1 Select AS
2 (optional) Retrieve AS Metadata
3 Pushed Authorization Request/Response
4 Redirect to AS
5 Follow redirect: Authorization Request
6 User authentication and consent (out of scope)
7 Redirect to client: Authorization Response
8 Follow redirect
9 Token Request/Response
Resource Server
10 Log in user
11 Resource Request/Response

Why not just use “vanilla” DY model instead of WIM?

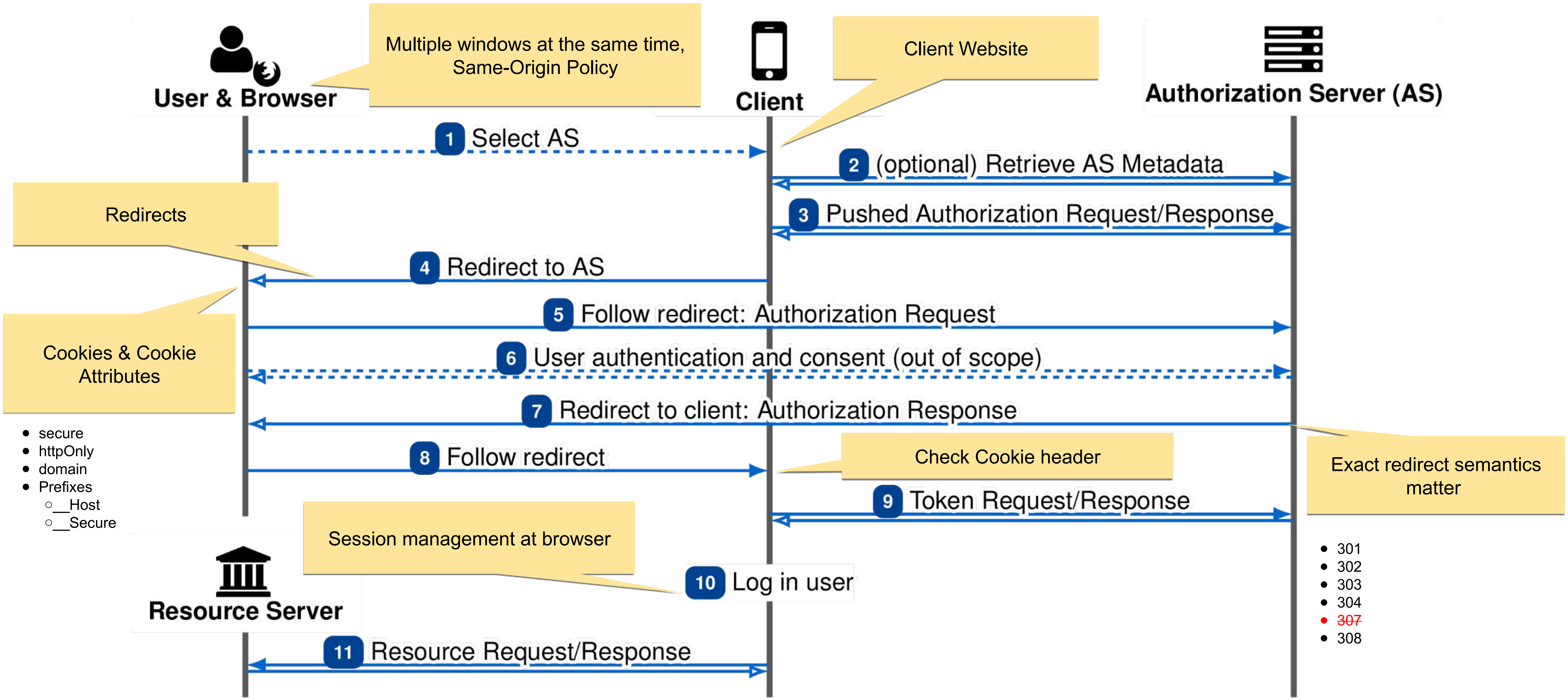

Multiple windows at the same time,
Same-Origin Policy
Client Website
User & Browser
Client
Authorization Server (AS)
1 Select AS
2 (optional) Retrieve AS Metadata
3 Pushed Authorization Request/Response
4 Redirect to AS
5 Follow redirect: Authorization Request
6 User authentication and consent (out of scope)
7 Redirect to client: Authorization Response
8 Follow redirect
9 Token Request/Response
Resource Server
10 Log in user
11 Resource Request/Response

Why not just use "vanilla" DY model instead of WIM?


Multiple windows at the same time, Same-Origin Policy
Client Website
User & Browser
Client
Authorization Server (AS)
Redirects
1 Select AS
2 (optional) Retrieve AS Metadata
3 Pushed Authorization Request/Response
4 Redirect to AS
5 Follow redirect: Authorization Request
6 User authentication and consent (out of scope)
7 Redirect to client: Authorization Response
8 Follow redirect
9 Token Request/Response
Exact redirect semantics matter
Resource Server
10 Log in user
11 Resource Request/Response

- 301
- 302
- 303
- 304
- ~~307~~
- 308

Why not just use "vanilla" DY model instead of WIM?

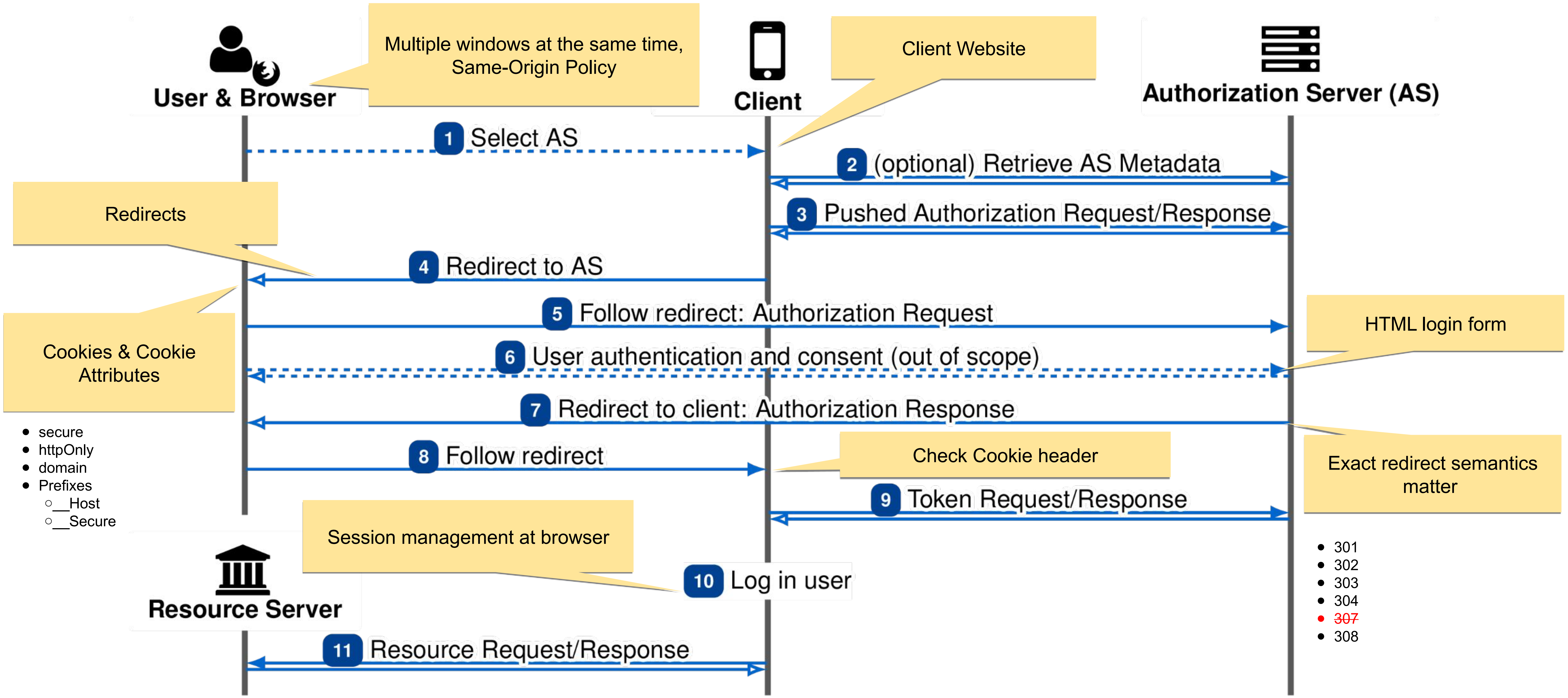


Multiple windows at the same time, Same-Origin Policy
Client Website
User & Browser
Client
Authorization Server (AS)
1 Select AS
2 (optional) Retrieve AS Metadata
3 Pushed Authorization Request/Response
Redirects
4 Redirect to AS
5 Follow redirect: Authorization Request
Cookies & Cookie Attributes
6 User authentication and consent (out of scope)
7 Redirect to client: Authorization Response
8 Follow redirect
Check Cookie header
Exact redirect semantics matter
9 Token Request/Response
Session management at browser
10 Log in user
Resource Server
11 Resource Request/Response

- secure
- httpOnly
- domain
- Prefixes
  - __Host
  - __Secure

- 301
- 302
- 303
- 304
- ~~307~~
- 308

Why not just use "vanilla" DY model instead of WIM?

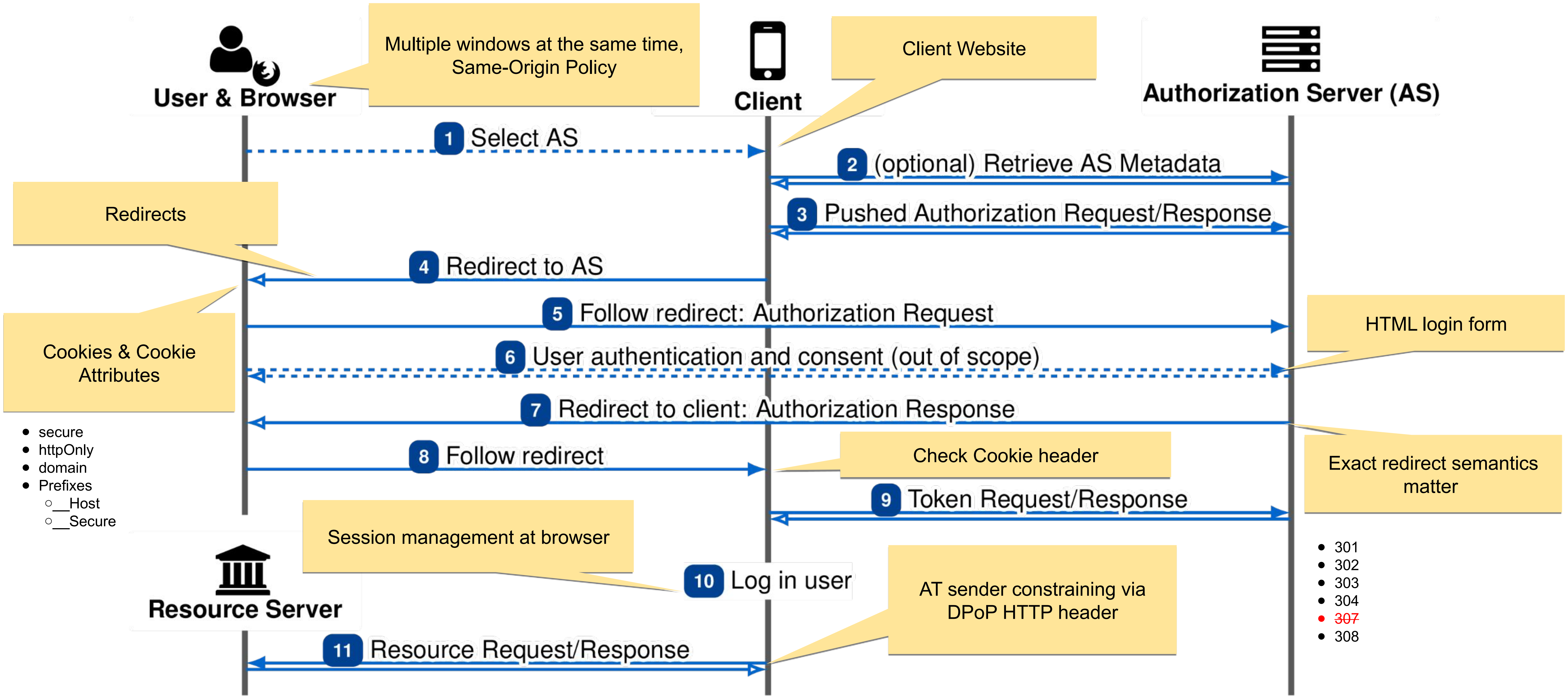


User & Browser
Multiple windows at the same time, Same-Origin Policy
Client
Client Website
Authorization Server (AS)
1 Select AS
2 (optional) Retrieve AS Metadata
Redirects
3 Pushed Authorization Request/Response
4 Redirect to AS
Cookies & Cookie Attributes
5 Follow redirect: Authorization Request
HTML login form
6 User authentication and consent (out of scope)
7 Redirect to client: Authorization Response
8 Follow redirect
Check Cookie header
Exact redirect semantics matter
9 Token Request/Response
Session management at browser
10 Log in user
Resource Server
11 Resource Request/Response

- secure
- httpOnly
- domain
- Prefixes
  - __Host
  - __Secure

- 301
- 302
- 303
- 304
- ~~307~~
- 308

Why not just use “vanilla” DY model instead of WIM?



User & Browser
Client
Authorization Server (AS)
Resource Server
1 Select AS
2 (optional) Retrieve AS Metadata
3 Pushed Authorization Request/Response
4 Redirect to AS
5 Follow redirect: Authorization Request
6 User authentication and consent (out of scope)
7 Redirect to client: Authorization Response
8 Follow redirect
9 Token Request/Response
10 Log in user
11 Resource Request/Response
Multiple windows at the same time, Same-Origin Policy
Client Website
Redirects
Cookies & Cookie Attributes
HTML login form
Check Cookie header
Exact redirect semantics matter
Session management at browser
AT sender constraining via DPoP HTTP header

- secure
- httpOnly
- domain
- Prefixes
  - __Host
  - __Secure

- 301
- 302
- 303
- 304
- ~~307~~
- 308

Why not just use “vanilla” DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
  - Redirects (303 vs. 307, …)
  - Session Management (cookies & cookie attributes, same-origin policy, …)
  - Multiple documents at the same time (attacker scripts along honest sites, …)
  - HTTP headers (Location, Authorization, DPoP, Cookies, …)

Why not just use "vanilla" DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
  - Redirects (303 vs. 307, …)
  - Session Management (cookies & cookie attributes, same-origin policy, …)
  - Multiple documents at the same time (attacker scripts along honest sites, …)
  - HTTP headers (Location, Authorization, DPoP, Cookies, …)
- Much stronger security guarantees

Why not just use “vanilla” DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
  - Redirects (303 vs. 307, …)
  - Session Management (cookies & cookie attributes, same-origin policy, …)
  - Multiple documents at the same time (attacker scripts along honest sites, …)
  - HTTP headers (Location, Authorization, DPoP, Cookies, …)
- Much stronger security guarantees
  - All Web features may be valuable for attacks, not just the ones used by the specific protocol

Why not just use “vanilla” DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
  - Redirects (303 vs. 307, …)
  - Session Management (cookies & cookie attributes, same-origin policy, …)
  - Multiple documents at the same time (attacker scripts along honest sites, …)
  - HTTP headers (Location, Authorization, DPoP, Cookies, …)
- Much stronger security guarantees
  - All Web features may be valuable for attacks, not just the ones used by the specific protocol
  - Examples: Attacker could use

Why not just use “vanilla” DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
    - Redirects (303 vs. 307, …)
    - Session Management (cookies & cookie attributes, same-origin policy, …)
    - Multiple documents at the same time (attacker scripts along honest sites, …)
    - HTTP headers (Location, Authorization, DPoP, Cookies, …)

- Much stronger security guarantees
    - All Web features may be valuable for attacks, not just the ones used by the specific protocol
    - Examples: Attacker could use
        - send cross-site requests via the honest browser (CSRF attacks)

Why not just use “vanilla” DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
  - Redirects (303 vs. 307, …)
  - Session Management (cookies & cookie attributes, same-origin policy, …)
  - Multiple documents at the same time (attacker scripts along honest sites, …)
  - HTTP headers (Location, Authorization, DPoP, Cookies, …)

- Much stronger security guarantees
  - All Web features may be valuable for attacks, not just the ones used by the specific protocol
  - Examples: Attacker could use
    - send cross-site requests via the honest browser (CSRF attacks)
    - in-browser communication, e.g., postMessages between iframes [S&P’14, ESORICS’15]

Why not just use “vanilla” DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
  - Redirects (303 vs. 307, …)
  - Session Management (cookies & cookie attributes, same-origin policy, …)
  - Multiple documents at the same time (attacker scripts along honest sites, …)
  - HTTP headers (Location, Authorization, DPoP, Cookies, …)

- Much stronger security guarantees
  - All Web features may be valuable for attacks, not just the ones used by the specific protocol
  - Examples: Attacker could use
    - send cross-site requests via the honest browser (CSRF attacks)
    - in-browser communication, e.g., postMessages between iframes [S&P’14, ESORICS’15]
    - leakages, e.g., via Referer HTTP header [CCS’16]

Why not just use "vanilla" DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
    - Redirects (303 vs. 307, …)
    - Session Management (cookies & cookie attributes, same-origin policy, …)
    - Multiple documents at the same time (attacker scripts along honest sites, …)
    - HTTP headers (Location, Authorization, DPoP, Cookies, …)

- Much stronger security guarantees
    - All Web features may be valuable for attacks, not just the ones used by the specific protocol
    - Examples: Attacker could use
        - send cross-site requests via the honest browser (CSRF attacks)
        - in-browser communication, e.g., postMessages between iframes [S&P'14, ESORICS'15]
        - leakages, e.g., via Referer HTTP header [CCS'16]
        - vulnerabilities in browser script APIs [S&P'22]

Why not just use “vanilla” DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
  - Redirects (303 vs. 307, …)
  - Session Management (cookies & cookie attributes, same-origin policy, …)
  - Multiple documents at the same time (attacker scripts along honest sites, …)
  - HTTP headers (Location, Authorization, DPoP, Cookies, …)

- Much stronger security guarantees
  - All Web features may be valuable for attacks, not just the ones used by the specific protocol
  - Examples: Attacker could use
    - send cross-site requests via the honest browser (CSRF attacks)
    - in-browser communication, e.g., postMessages between iframes [S&P’14, ESORICS’15]
    - leakages, e.g., via Referer HTTP header [CCS’16]
    - vulnerabilities in browser script APIs [S&P’22]
    - malicious scripts interpreted by honest browser

Why not just use “vanilla” DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
  - Redirects (303 vs. 307, …)
  - Session Management (cookies & cookie attributes, same-origin policy, …)
  - Multiple documents at the same time (attacker scripts along honest sites, …)
  - HTTP headers (Location, Authorization, DPoP, Cookies, …)

- Much stronger security guarantees
  - All Web features may be valuable for attacks, not just the ones used by the specific protocol
  - Examples: Attacker could use
    - send cross-site requests via the honest browser (CSRF attacks)
    - in-browser communication, e.g., postMessages between iframes [S&P’14, ESORICS’15]
    - leakages, e.g., via Referer HTTP header [CCS’16]
    - vulnerabilities in browser script APIs [S&P’22]
    - malicious scripts interpreted by honest browser
  - Simplified model cannot capture these potential attack vectors

Why not just use “vanilla” DY model instead of WIM?

- Functional reasons: Web SSO protocols use many Web features, e.g.
  - Redirects (303 vs. 307, …)
  - Session Management (cookies & cookie attributes, same-origin policy, …)
  - Multiple documents at the same time (attacker scripts along honest sites, …)
  - HTTP headers (Location, Authorization, DPoP, Cookies, …)

- Much stronger security guarantees
  - All Web features may be valuable for attacks, not just the ones used by the specific protocol
  - Examples: Attacker could use
    - send cross-site requests via the honest browser (CSRF attacks)
    - in-browser communication, e.g., postMessages between iframes [S&P’14, ESORICS’15]
    - leakages, e.g., via Referer HTTP header [CCS’16]
    - vulnerabilities in browser script APIs [S&P’22]
    - malicious scripts interpreted by honest browser
  - Simplified model cannot capture these potential attack vectors

Protocols/Standards We Have Analyzed So Far

- OAuth 2.0
- OpenID Connect
- OpenID FAPI 1.0 and FAPI 2.0
- OpenID Federation 1.0
- OpenID Connect Client-Initiated Backchannel Authentication Flow (CIBA)
- GNAP
- Mozilla BrowserID
- OID4VP/VCI (ongoing work)
- Web Payment APIs
- …

Audience Injection Attack

307 Redirect Attack

IdP Mix-Up Attack

State Leak Attack

Naive RP Session Integrity Attack

Access Token Injection

Cuckoo’s Token Attack

PKCE Chosen Challenge Attack

….

Protocols/Standards We Have Analyzed So Far

under submission: affects several standards related to OAuth 2.0, OpenID Connect, FAPI, CIBA, ...

- OAuth 2.0
- OpenID Connect
- OpenID FAPI 1.0 and FAPI 2.0
- OpenID Federation 1.0
- OpenID Connect Client-Initiated Backchannel Authentication Flow (CIBA)
- GNAP
- Mozilla BrowserID
- OID4VP/VCI (ongoing work)
- Web Payment APIs
- ...

Audience Injection Attack

307 Redirect Attack

IdP Mix-Up Attack

State Leak Attack

Naive RP Session Integrity Attack

Access Token Injection

Cuckoo's Token Attack

PKCE Chosen Challenge Attack

....

But this was not about breaking things ...

We always started out with

1) Modeling
2) Formalizing security properties
3) Trying to prove properties

Our findings resulted in fixed/improved and formally analyzed standards.

Close interaction with standardization bodies (IETF, OpenID Foundation, ...)

Audience Injection Attack

307 Redirect Attack

IdP Mix-Up Attack

State Leak Attack

Naive RP Session Integrity Attack

Access Token Injection

Cuckoo's Token Attack

PKCE Chosen Challenge Attack

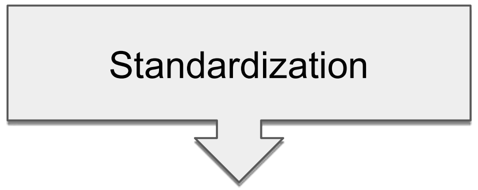
....

Modes of Operation with Standardization Bodies

This is how we started:

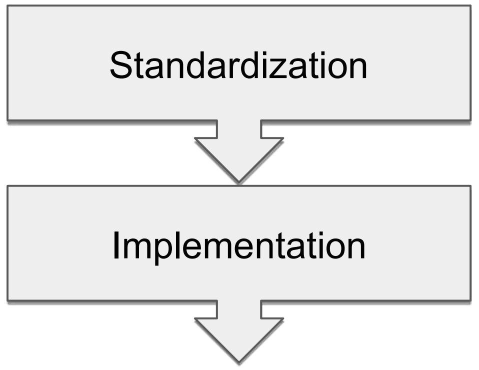
Modes of Operation with Standardization Bodies

This is how we started:



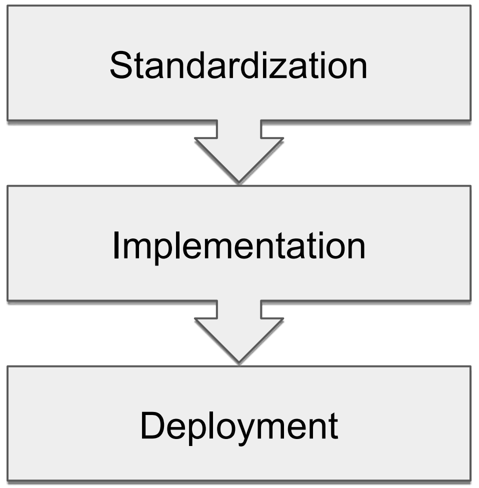
Modes of Operation with Standardization Bodies

This is how we started:



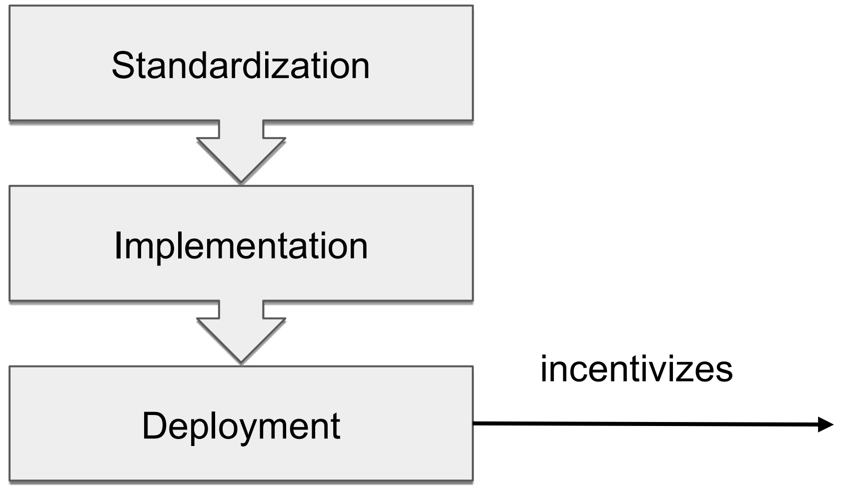
Modes of Operation with Standardization Bodies

This is how we started:



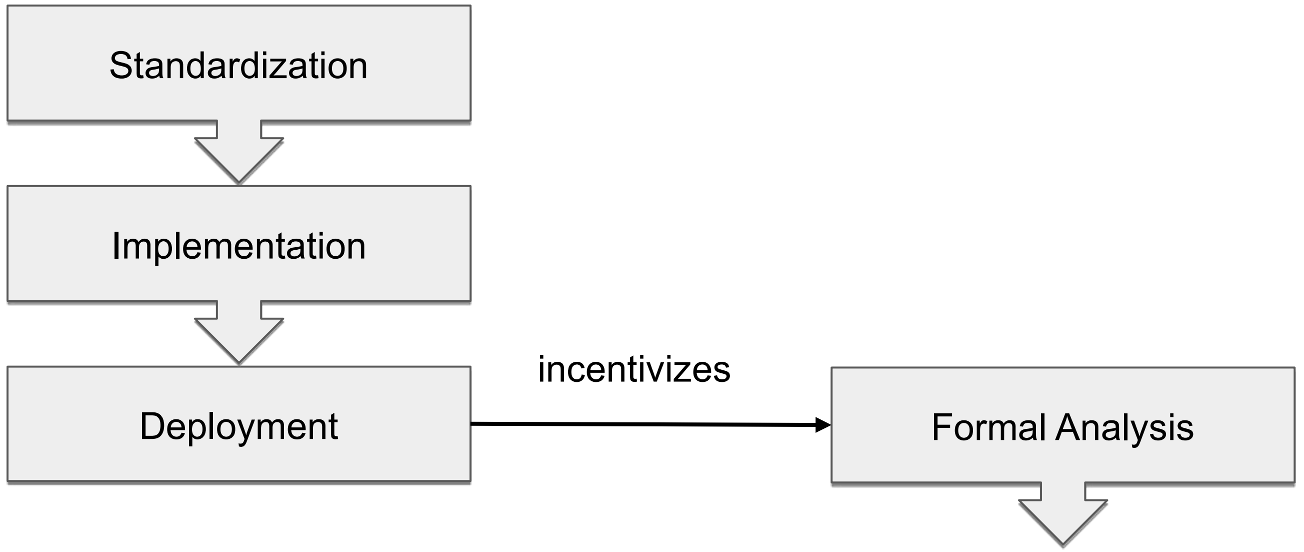
Modes of Operation with Standardization Bodies

This is how we started:



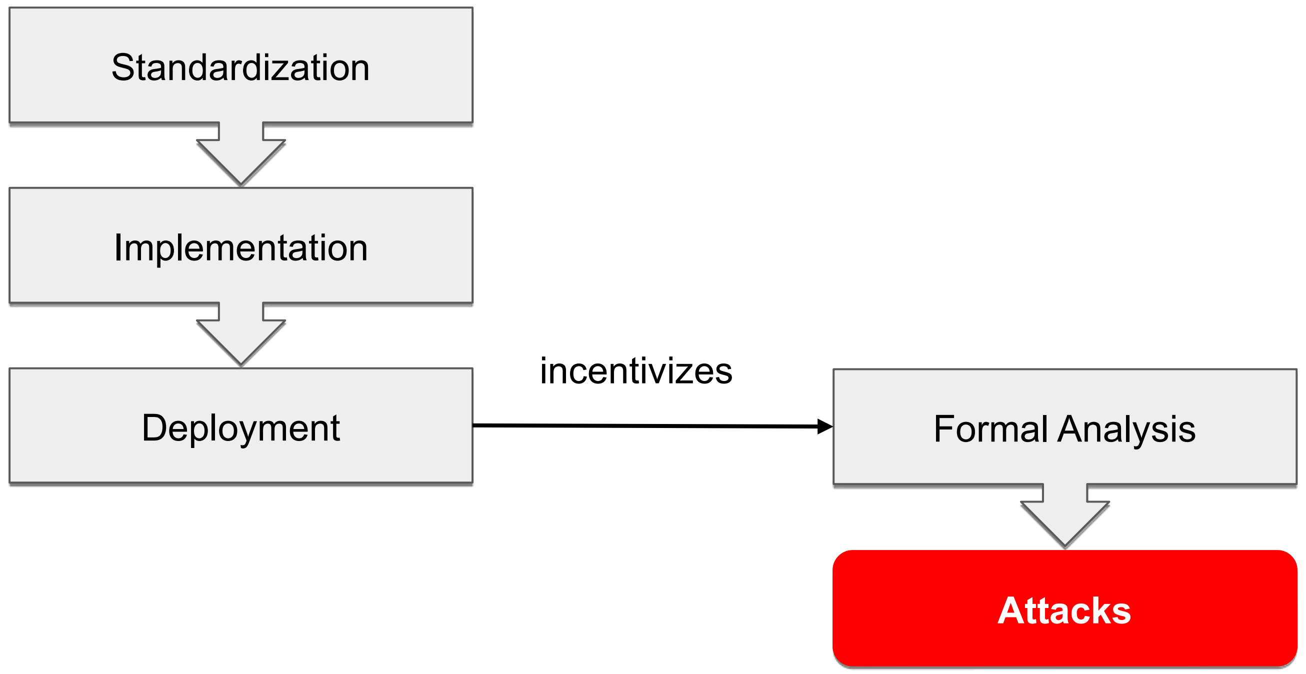
Modes of Operation with Standardization Bodies

This is how we started:



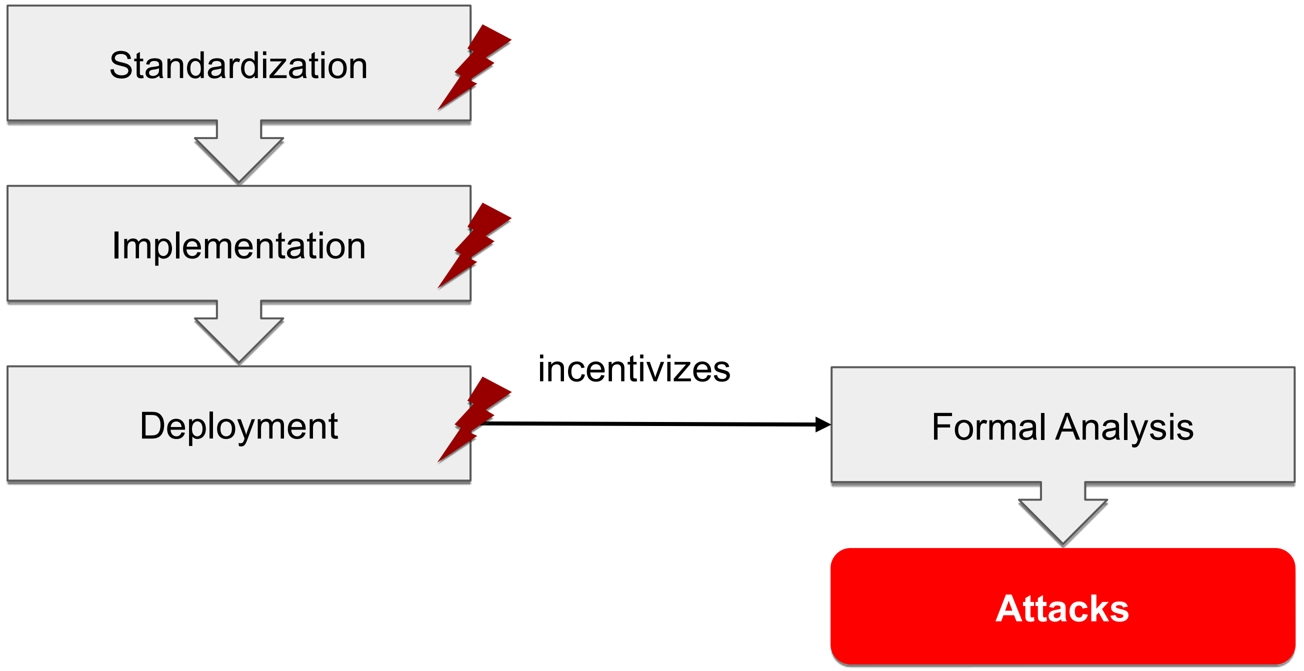
Modes of Operation with Standardization Bodies

This is how we started:



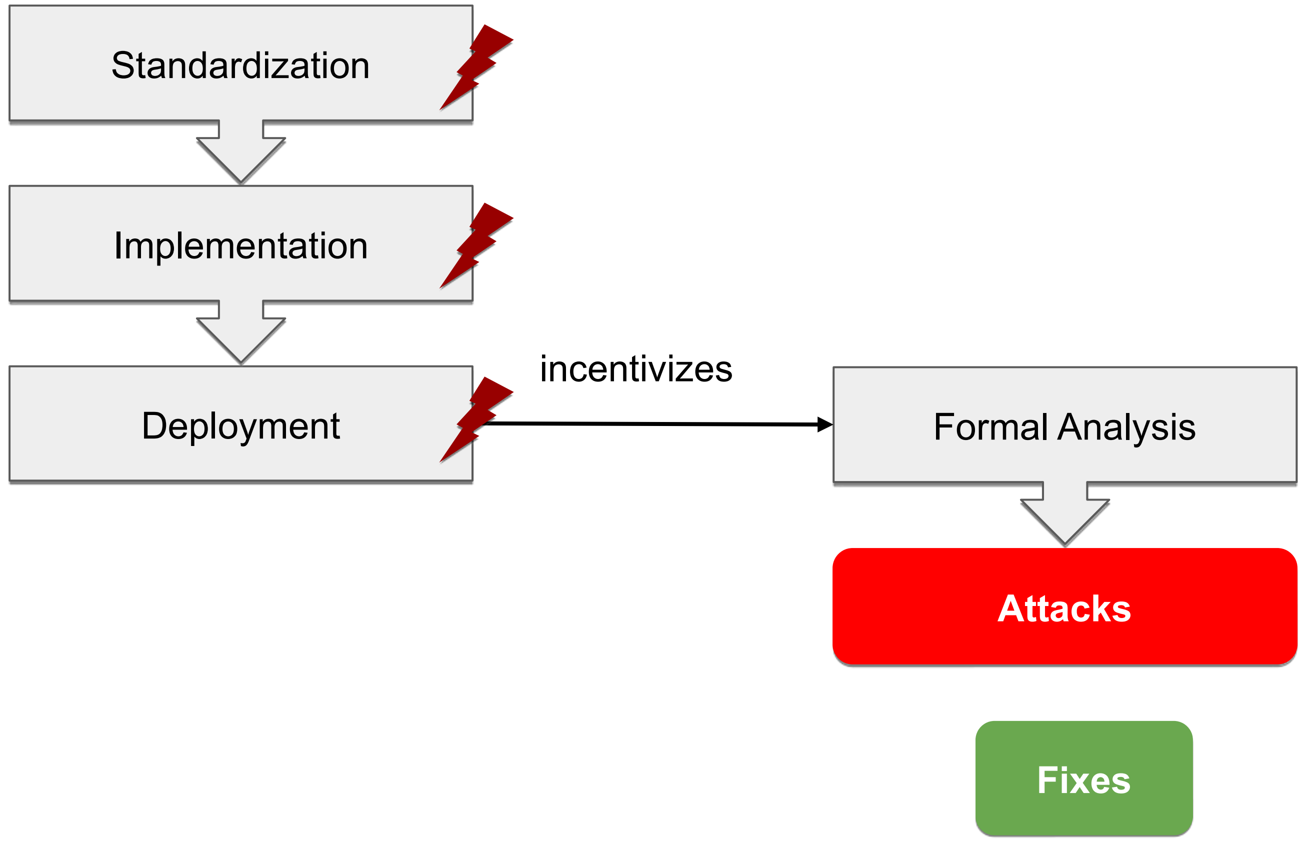
Modes of Operation with Standardization Bodies

This is how we started:



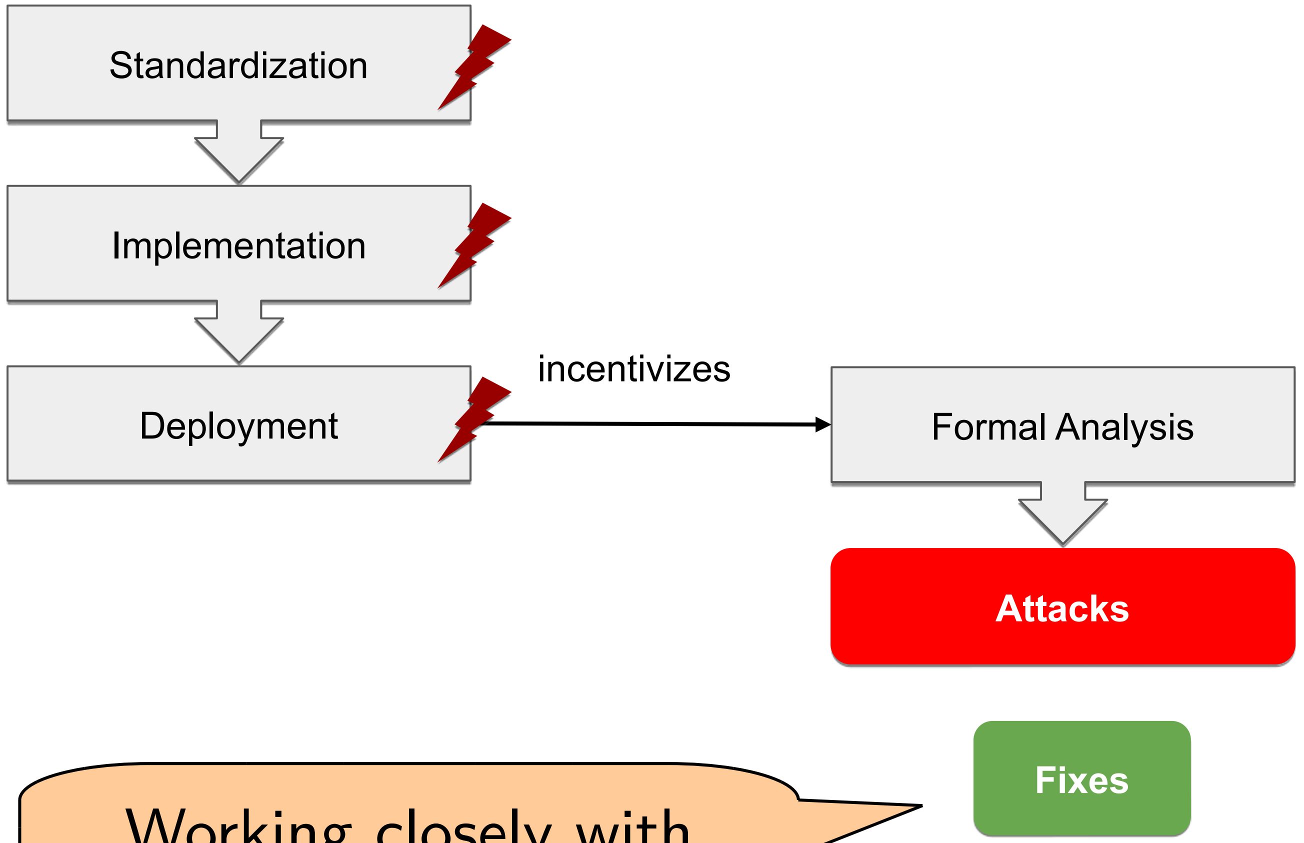
Modes of Operation with Standardization Bodies

This is how we started:



Modes of Operation with Standardization Bodies

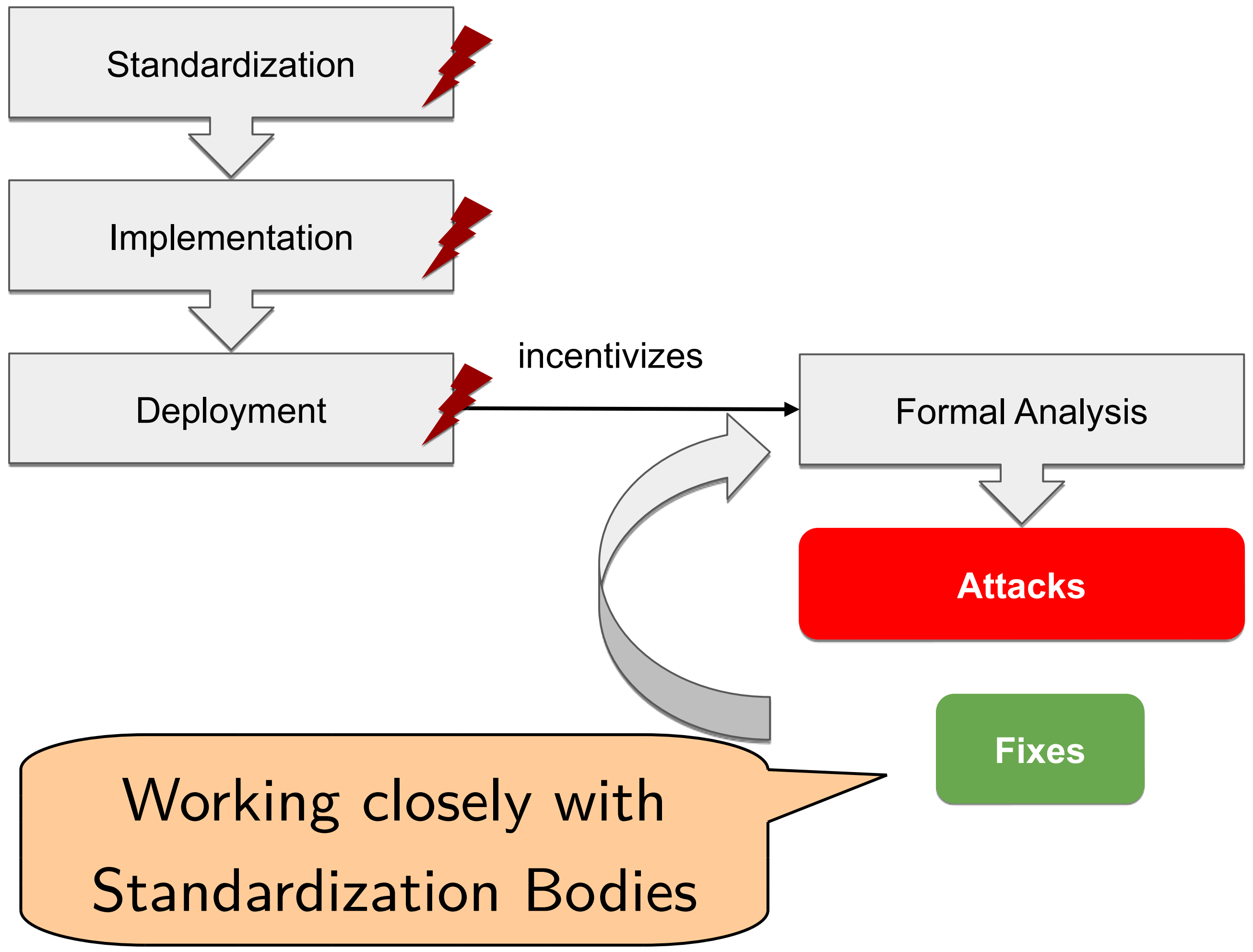
This is how we started:



Working closely with Standardization Bodies

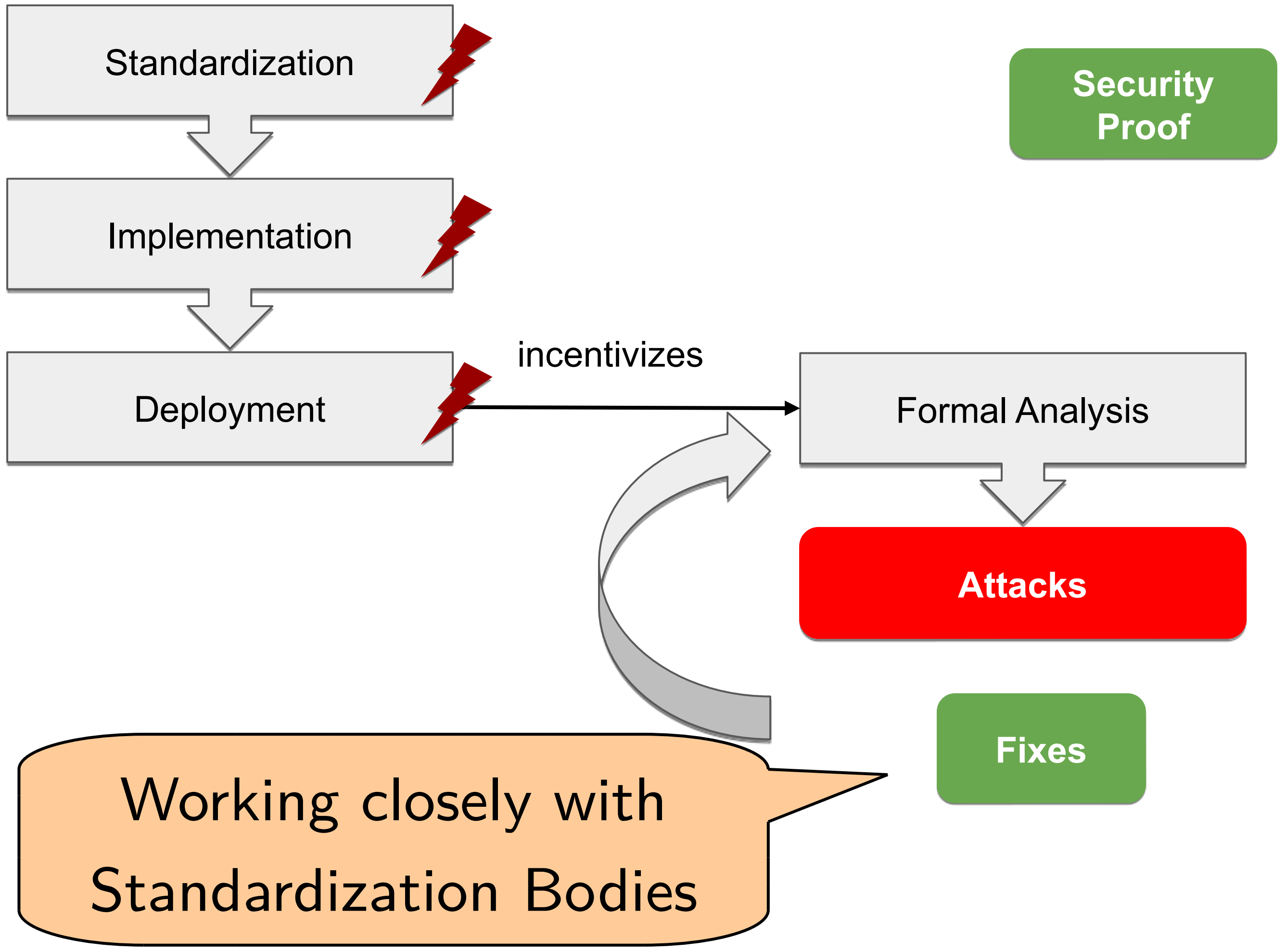
Modes of Operation with Standardization Bodies

This is how we started:



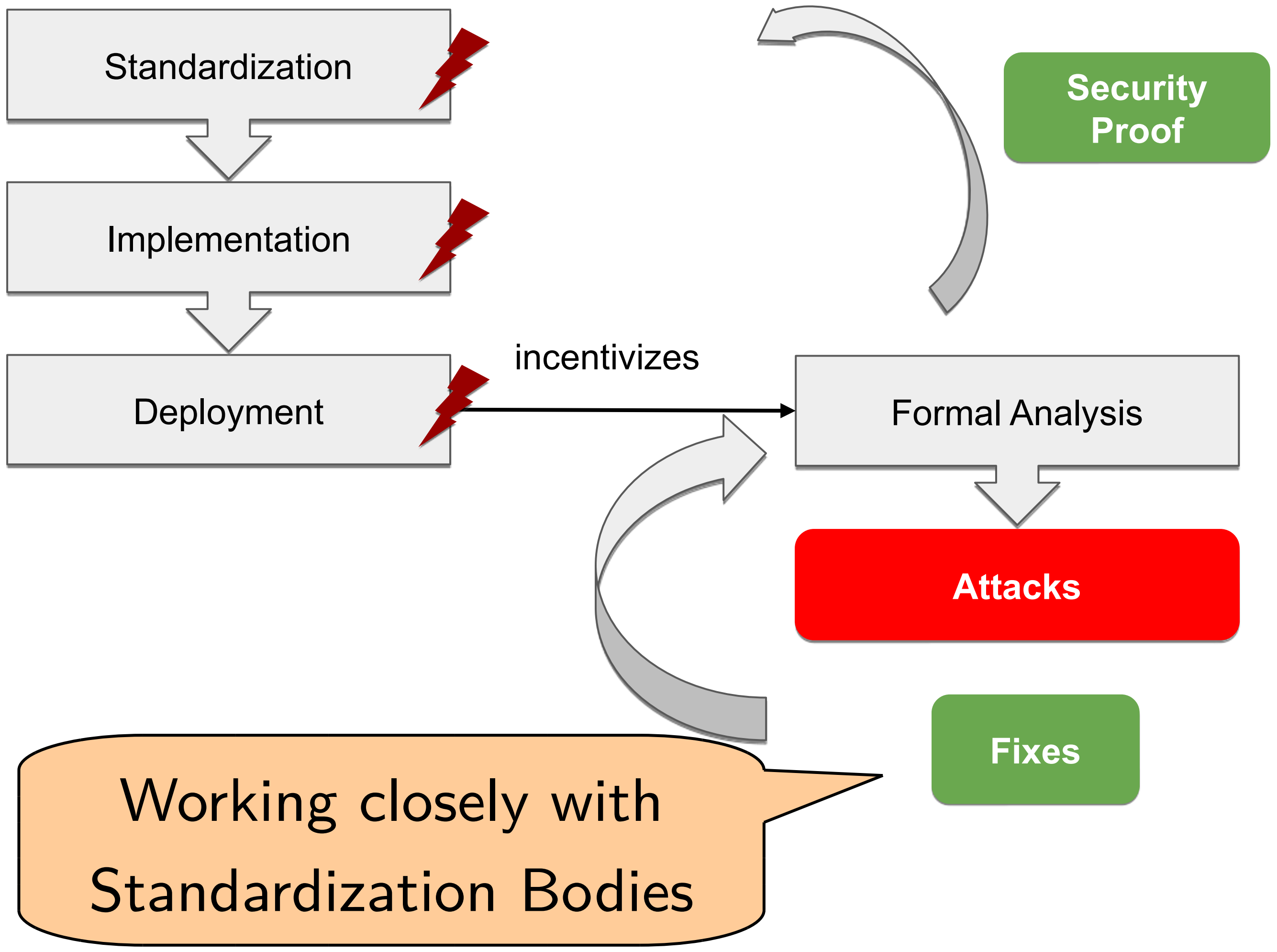
Modes of Operation with Standardization Bodies

This is how we started:



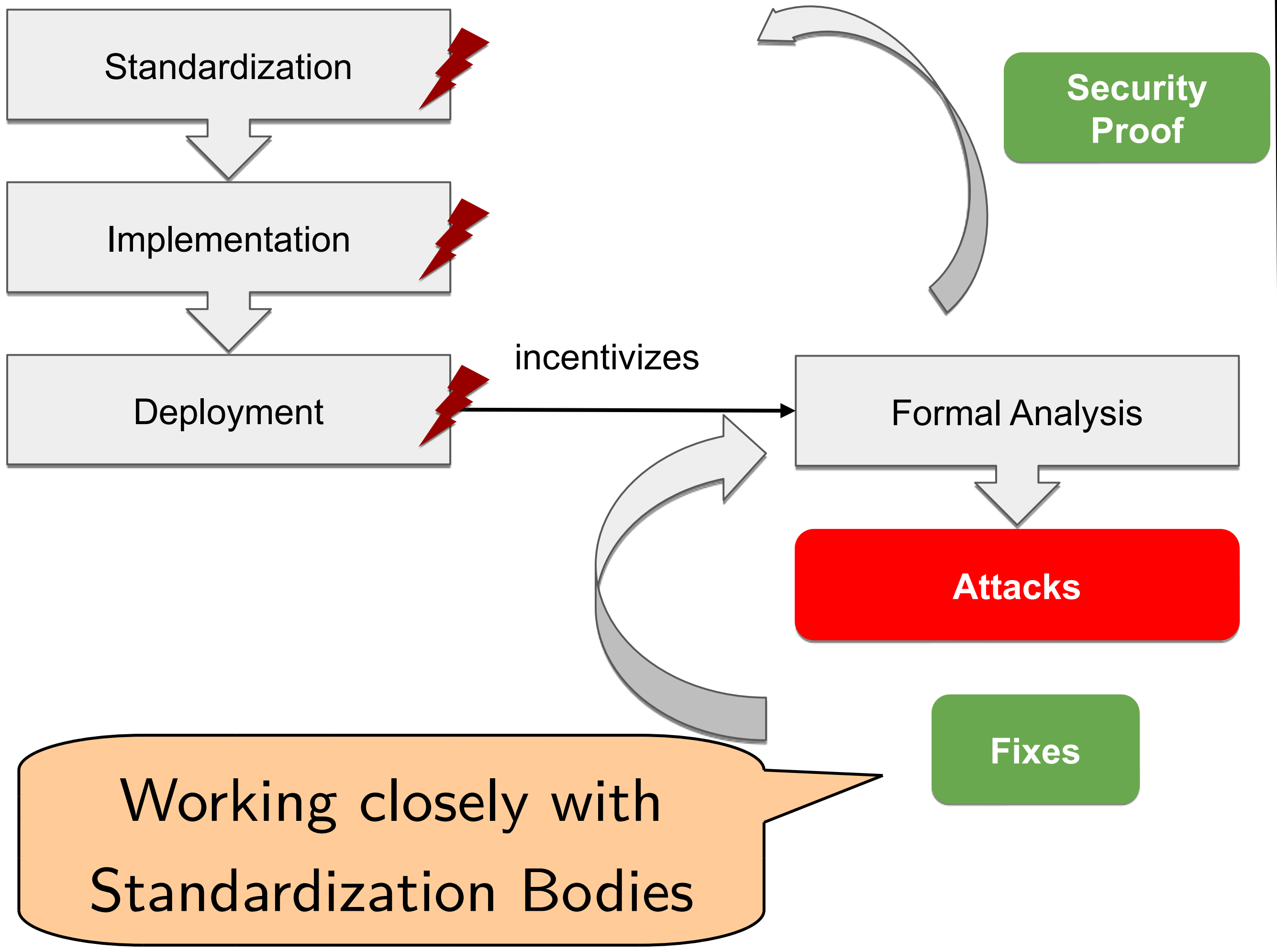
Modes of Operation with Standardization Bodies

This is how we started:

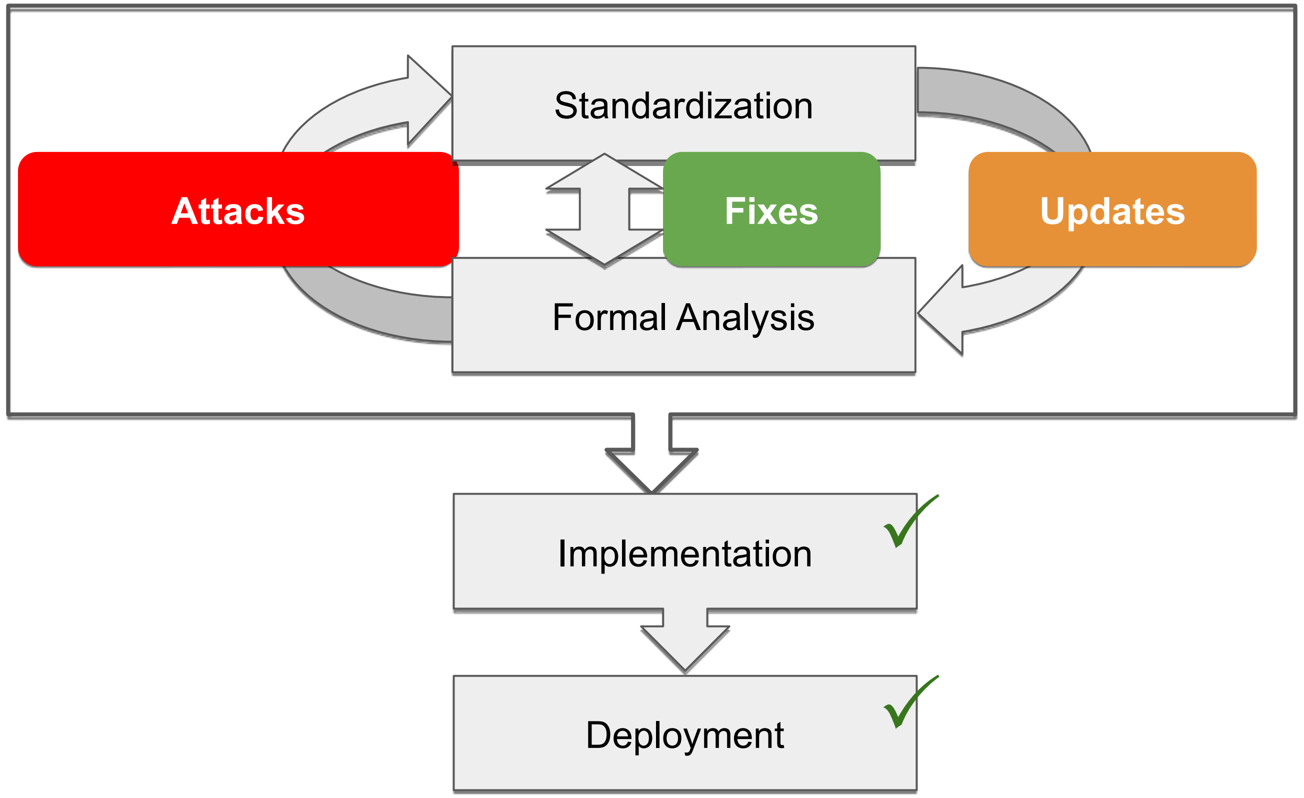


Modes of Operation with Standardization Bodies

This is how we started:



Now we are often part of the standardization process (OpenID Foundation, IETF):

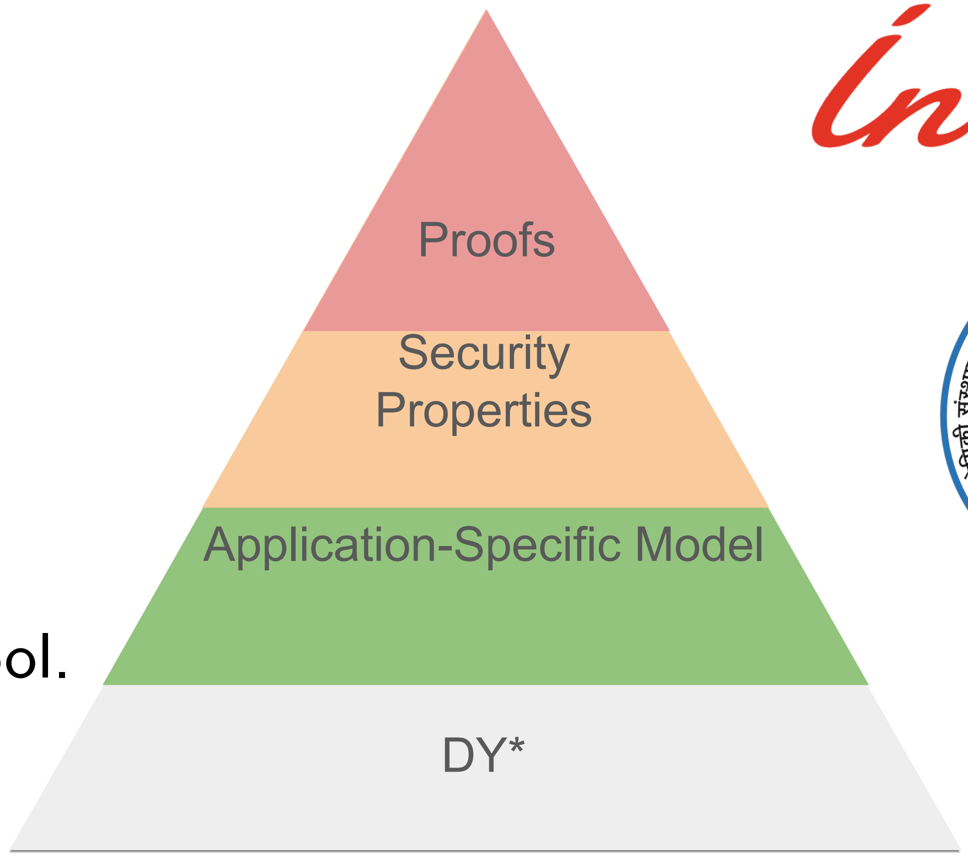


Towards Mechanizing the WIM

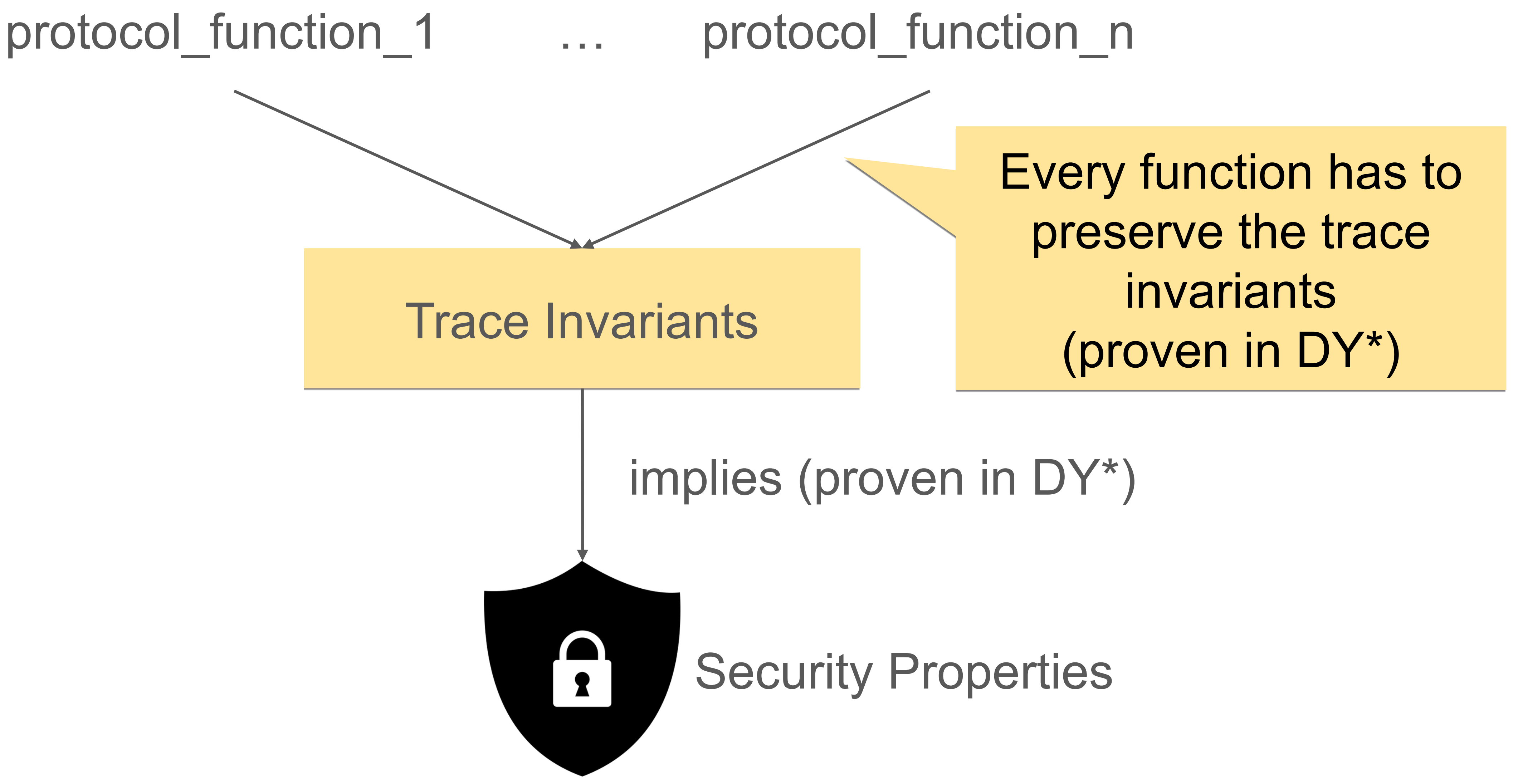
DY*

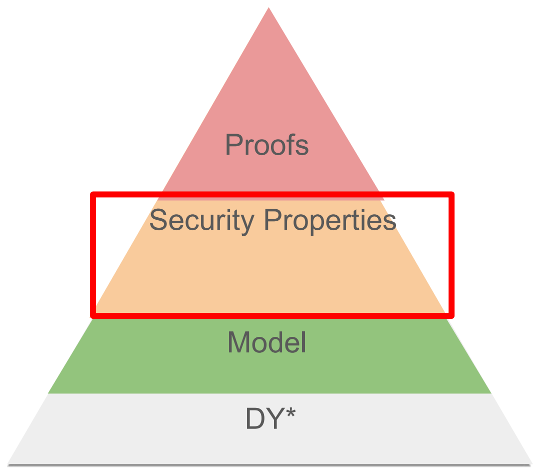
- Dolev-Yao model implemented in F*
- Enables fine-grained analysis up to implementation level
- Mechanized (tool checked) proofs
- Partially automated proofs
- Executable models
- Highly modular

At this point, general crypto protocol analysis tool.



Security Properties





Case Studies So Far

- Signal Messaging Protocol
  - Unbound number of rounds (ratcheting)
  - Forward Secrecy & Post Compromise Security
- Automatic Certificate Management Environment (ACME)
  - One of the largest & most in-depth formal security analyses in the literature (16.000 LoC)
  - ACME client model can interoperate with real-world server
- Needham-Schroeder(-Lowe), ISO-DH, and ISO-KEM

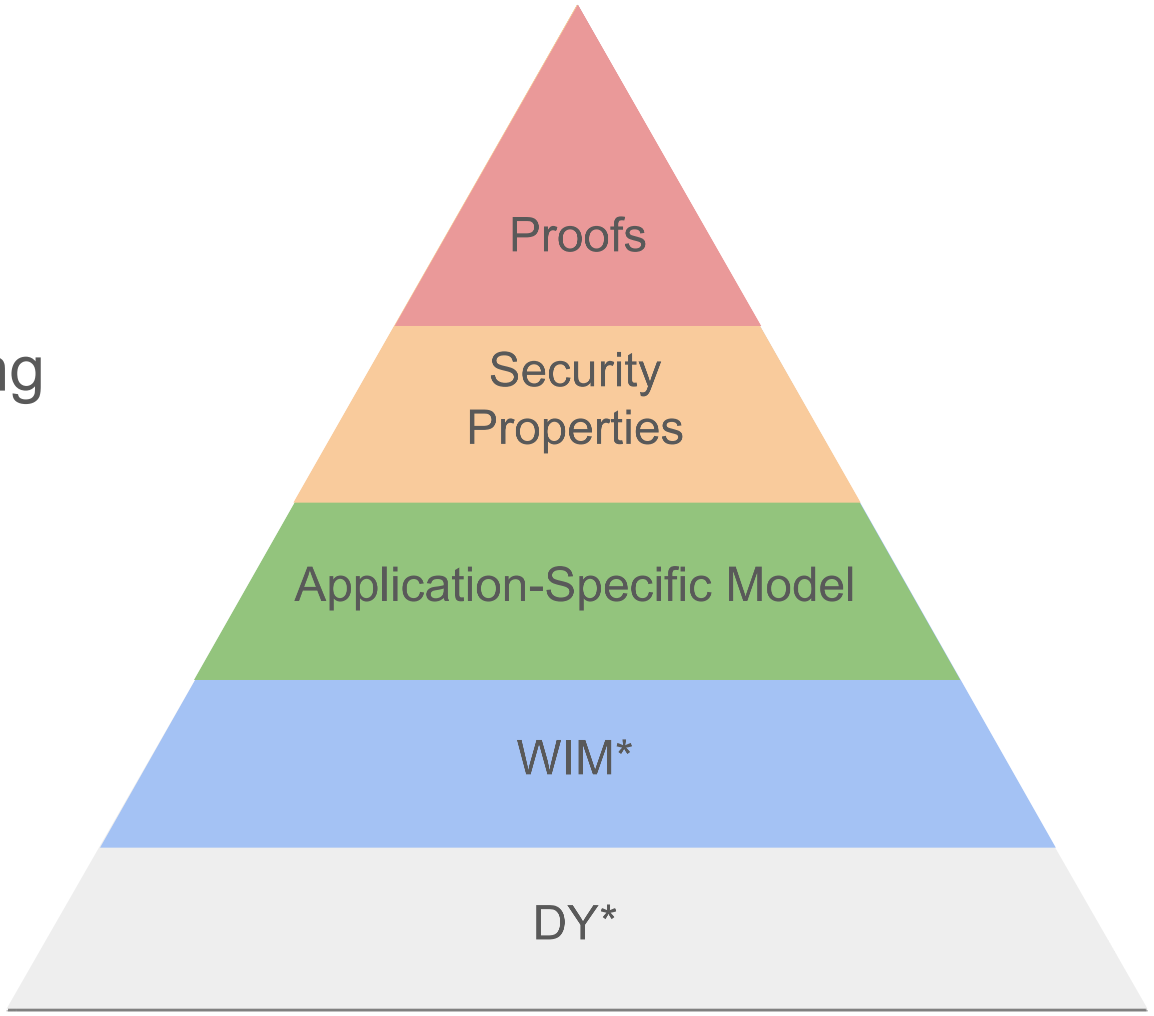
DY* Current/Future Work

Near-term

- Improving proof automation
- Database library
- HTTP library for sending requests and receiving responses

Long-term

- WIM*
    - Generic web server
    - Browser



Conclusion

Conclusion

- SSO protocols and standards are fun!
- The WIM is the most comprehensive model of the web infrastructure to date
- And has proved to be instrumental for formal analysis
- Several standards analyzed based on the WIM
- (Almost always) found new attacks and/or attack classes
- Proposed fixes
- Proved fixed standards secure in the WIM
  (under precisely formulated assumptions)
- Direct impact on standards
- Close collaboration with standardization bodies
- By now often involved in standardization process.

Conclusion

- SSO protocols and standards are fun!
- The WIM is the most comprehensive model of the web infrastructure to date
- And has proved to be instrumental for formal analysis
- Several standards analyzed based on the WIM
- (Almost always) found new attacks and/or attack classes
- Proposed fixes
- Proved fixed standards secure in the WIM
  (under precisely formulated assumptions)
- Direct impact on standards
- Close collaboration with standardization bodies
- By now often involved in standardization process.

Overview of formal methods for web security: Michele Bugliesi, Stefano Calzavara, Riccardo Focardi