

Single Sign On Inter-Enterprise

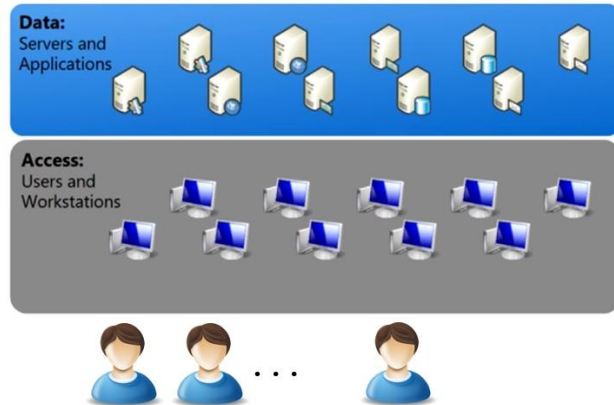


Identity and Access Management (IAM) (REMIND)



- ❑ **Procedures** and **technologies** for management of individual **identities**, their **authentication**, **authorization**, and **access rights**
- ❑ **within** or **across** **enterprise** boundaries

INTRA-Enterprise: AuthDB



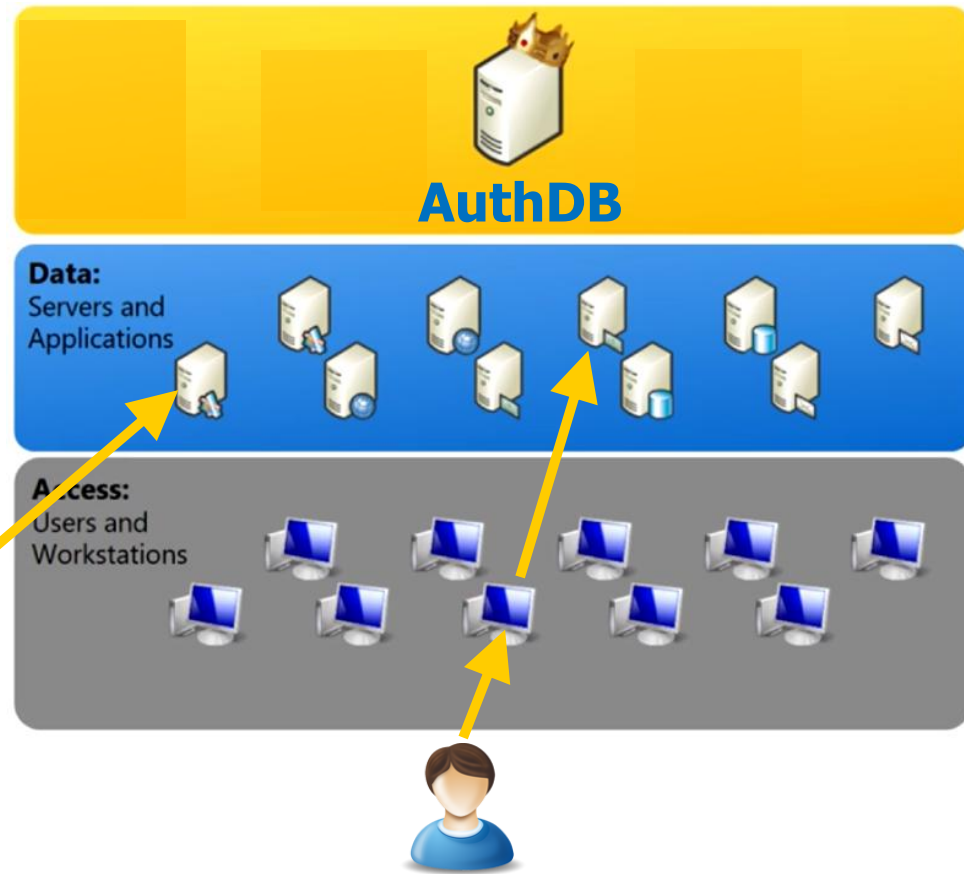
DIRECTORY SERVICE



- ❑ **Centralized** repository (**Directory Service**) describes:
 - ❑ All **accounts** (including their **credentials**)
 - ❑ All **resources**
 - ❑ All **access rights** of accounts to resources (ACLs)

Single Sign On (SSO) INTRA-Enterprise

- ❑ **Accounts and Credentials** stored in DS
- ❑ Valid **everywhere**
- ❑ **Every authentication** involves DS
- ❑ Several possible implementations

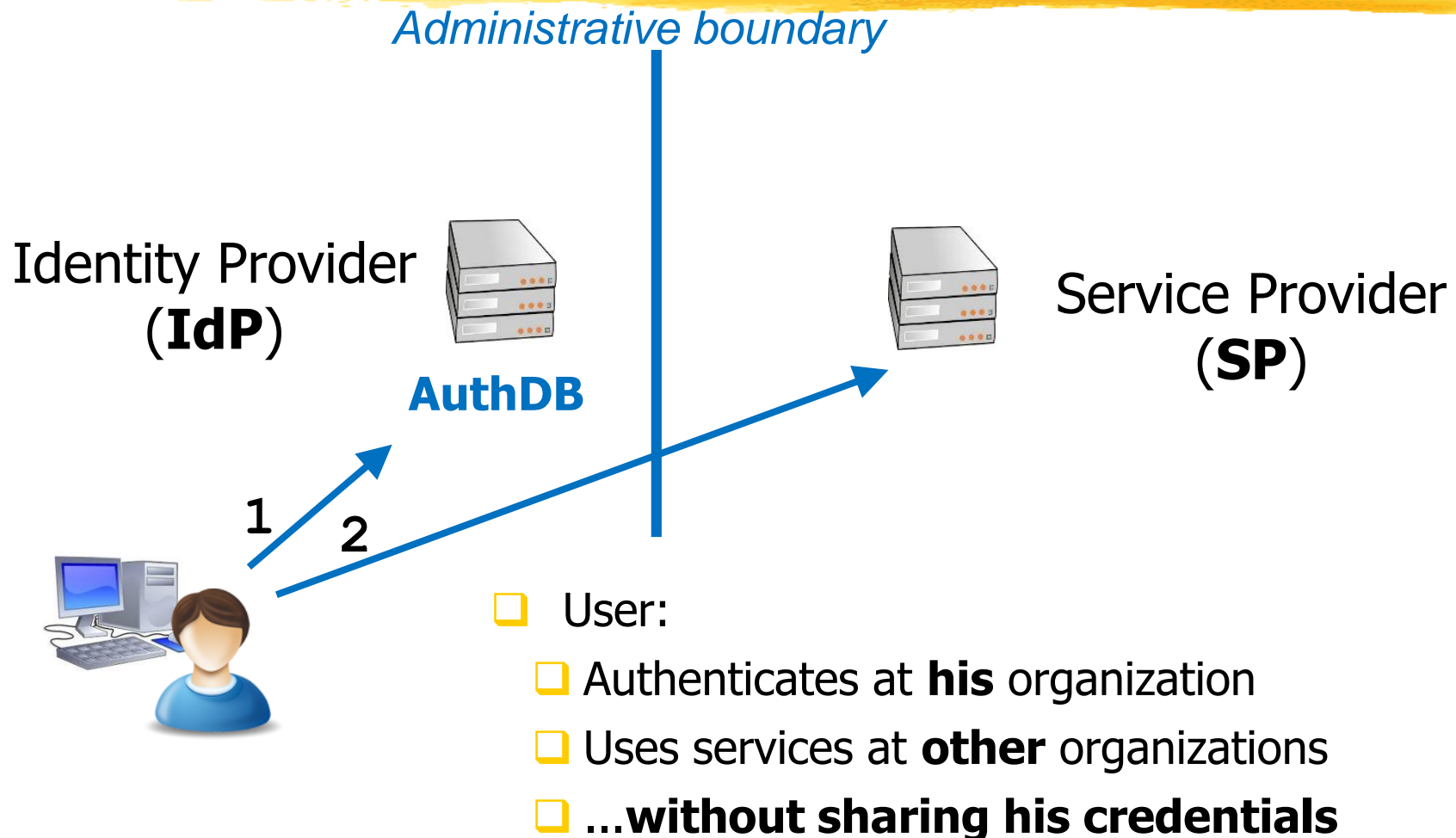


INTRA-Enterprise



- ❑ Widely prevalent technology:
 - ❑ **Windows Active Directory**
 - ❑ **Domain** \approx All IT entities in an organization
 - ❑ **Domain Controller** \approx Directory Service

SSO INTER-enterprise



Important Use Case



- ❑ Organization with IdP wants to **outsource** services **while keeping control** on:
 - ❑ **Authentication**
 - ❑ Credential lifecycle (issuance, revocation, expiration)
 - ❑ **Authorization**

Why it is not trivial

Administrative boundary

Service Provider
(SP)



Is that true?



"I am **potus**@**whitehouse.gov**"

"I am **5943**@**units.it**"

INTER-Enterprise (I)



- ❑ MANY technologies
- ❑ **Standard** for **web-based** interactions (HTTPS)
 - ❑ **OAuth**
 - ❑ Authentication **and** Authorization
 - ❑ **OpenID Connect** (based on OAuth)
 - ❑ Authentication
 - ❑ **SAML** \approx OAuth

INTER-Enterprise (II)



- ❑ MANY technologies
- ❑ **Standard** for web-based interactions (HTTPS)
- ❑ Many **implementations** of these standards
 - ❑ Google, Amazon, Microsoft, Okta, KeyCloak, ...
- ❑ In principle, **interoperable**
 - ❑ IdP impl-X \leftrightarrow SP impl-Y
- ❑ In practice, ahem...

INTER-Enterprise (III)

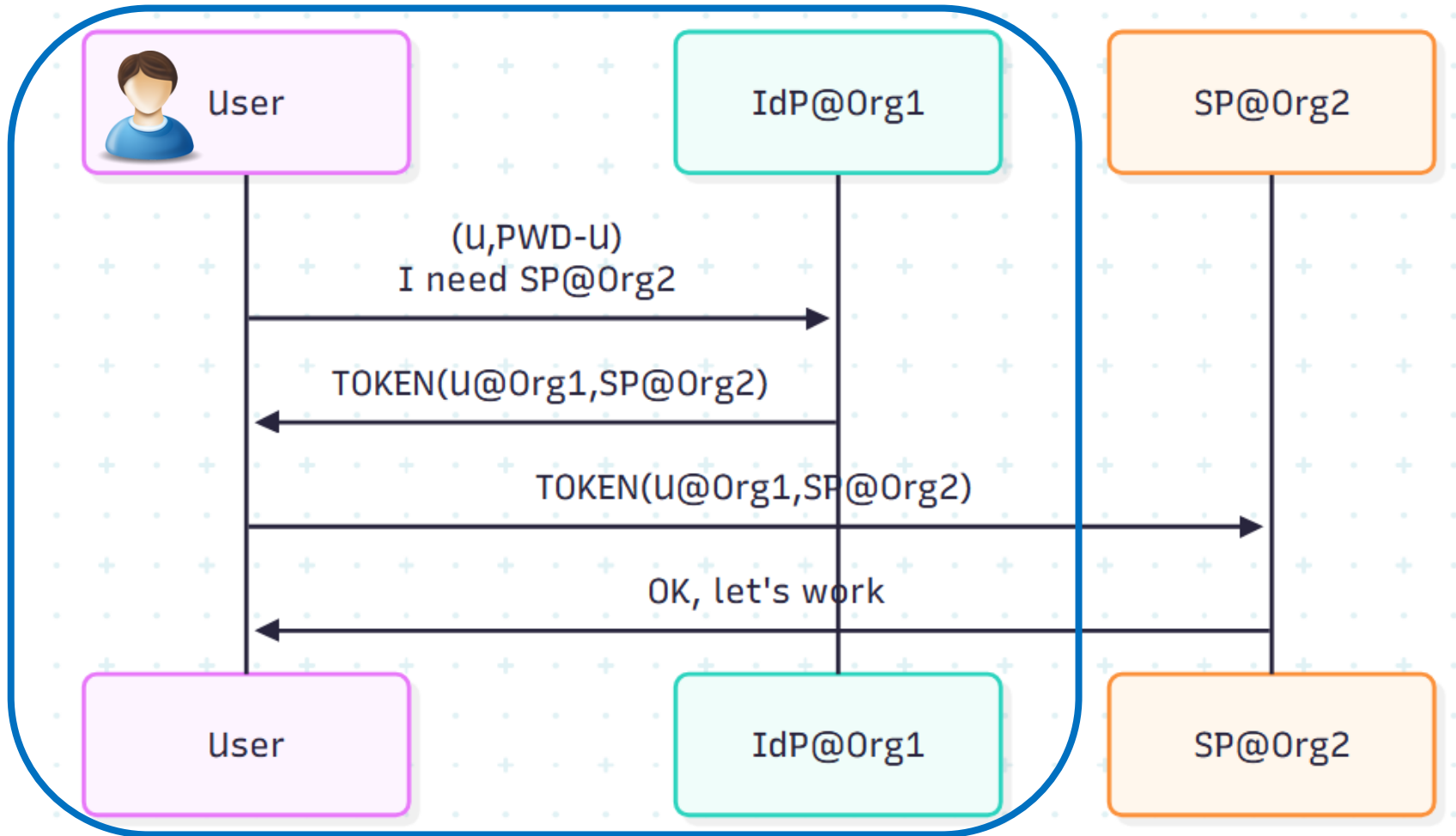


- ❑ MANY technologies
- ❑ **Standard** for web-based interactions (HTTPS)
- ❑ Technologies for "**any protocol**"
 - ❑ Kerberos Realms
 - ❑ DC@Org-A issues a ticket for SP@Org-B
 - ❑ Microsoft Entra
 - ❑ Web-IdP → Internal-SP

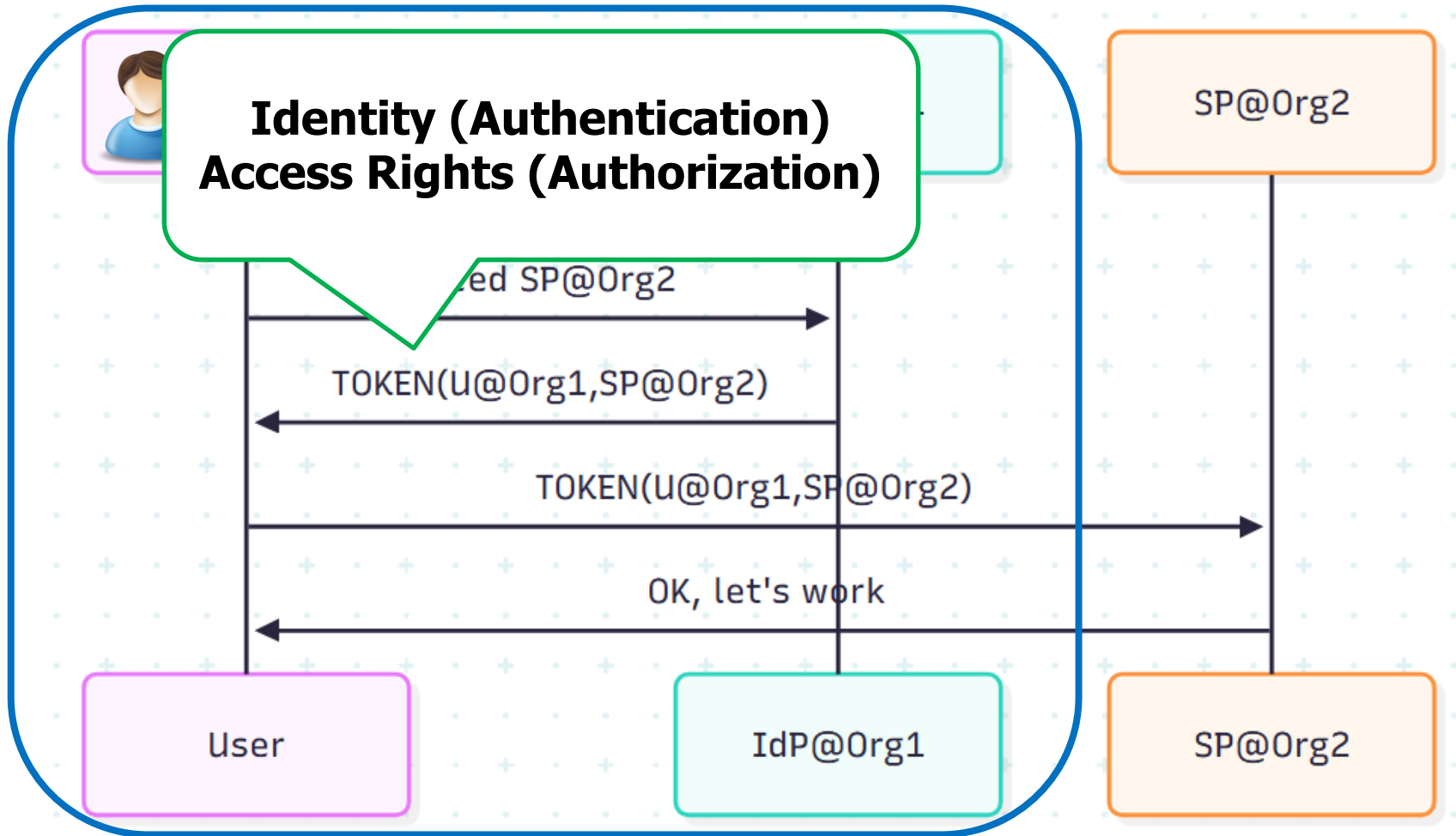
Implementation: What is needed?



SSO INTER-enterprise: Basic Idea (I)



SSO INTER-enterprise: Basic Idea (II)



What we need (I)

TOKEN(U@Org1,SP@Org2)

SP@Org2

SP@Org2

- ☐ **Authentic?**
- ☐ **Intact?**



What we need (II)

TOKEN(U@Org1,SP@Org2)

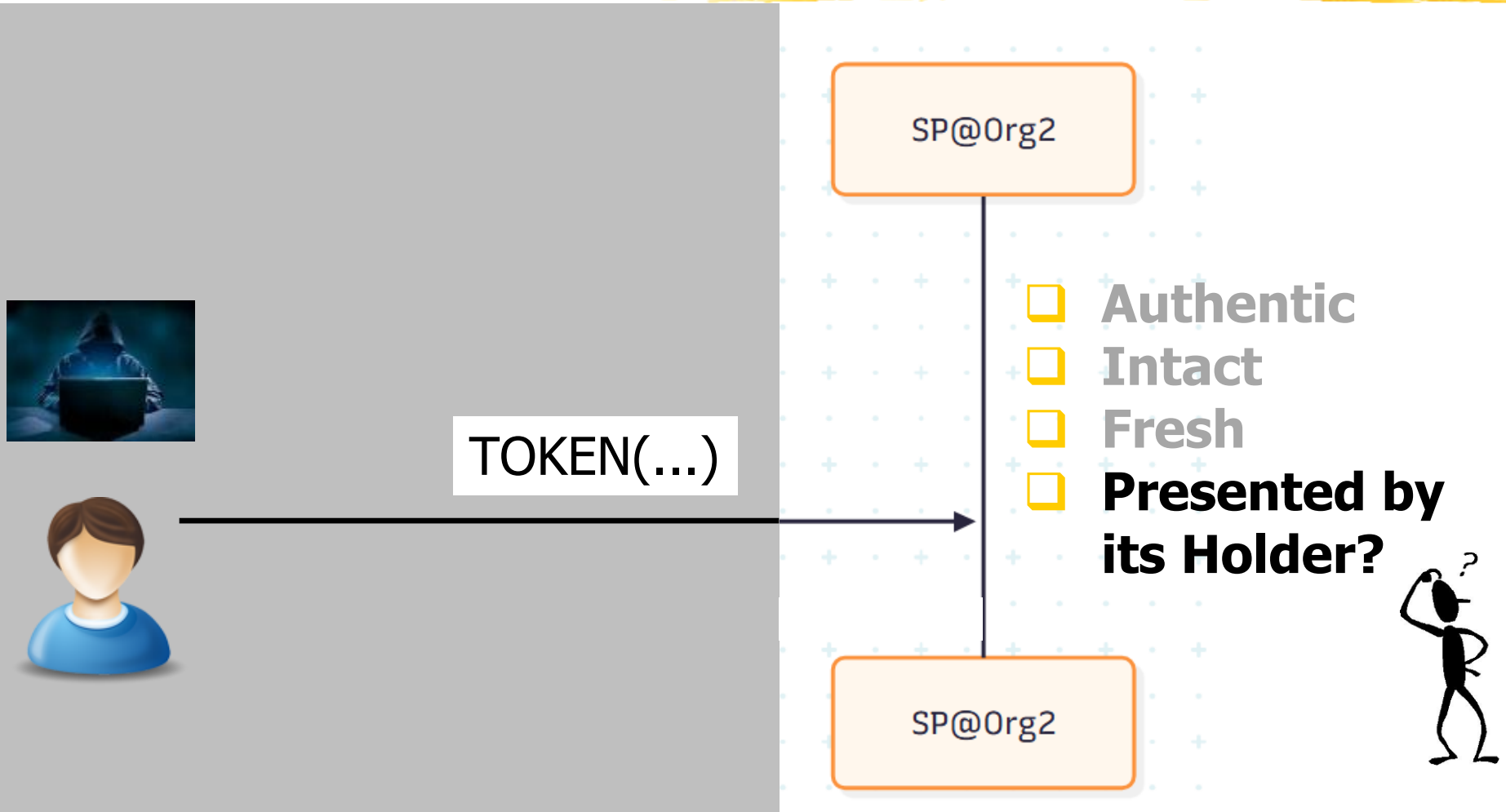
SP@Org2

SP@Org2

- ☐ Authentic
- ☐ Intact
- ☐ Fresh?



What we need (III)



What we have



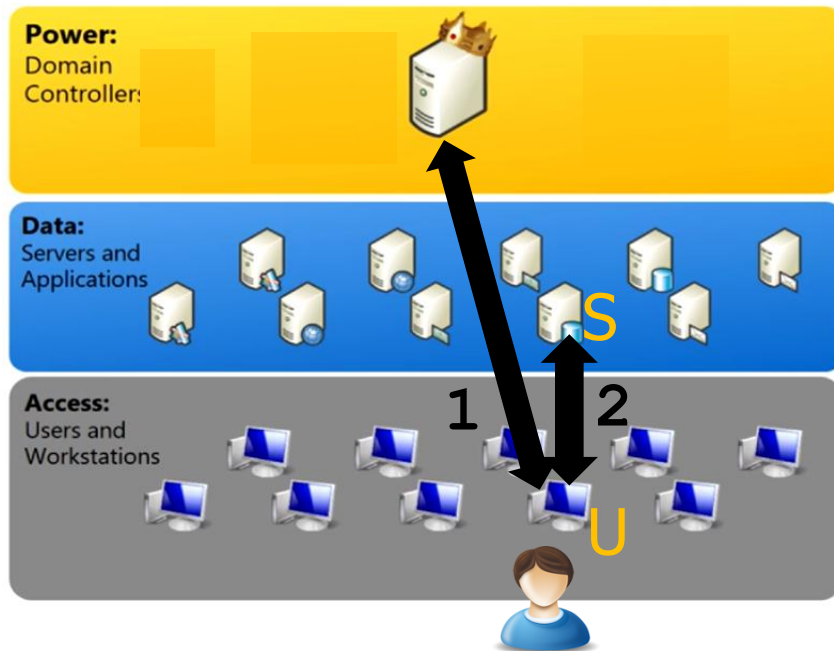
- ❑ Ability to transfer token T from IdP to SP across an **untrusted** channel
 - ❑ SP can verify **authenticity, integrity, freshness** of T
 - ❑ SP can verify that T is **presented by its owner**
- ❑ MANY standards/technologies for providing these capabilities

Remark 1



- ❑ Ability to transfer token T from DC to S across an **untrusted** channel
 - ❑ S can verify **authenticity, integrity, freshness** of T
 - ❑ S can verify that T is presented by **its owner**
- ❑ **INTRA**-Enterprise is the **very same** scenario
- ❑ Main solution: Kerberos

Remark 1 - Basic Idea



S needs to:

1. **Authenticate U**
2. Make sure U has **access rights** for network logon on S

Usually: **Kerberos**

- ❑ $ST(U,S)$ = Service Ticket
- ❑ Can be used only by U for accessing S
- ❑ Contains access rights

Remark 2



- ❑ Ability to transfer token T from IdP/DC to SP/S across an **untrusted** channel
 - ❑ S can verify **authenticity, integrity, freshness** of T
 - ❑ S can verify that T is presented by **its owner**

- ❑ Building **centralized authorization** with these capabilities is (conceptually) simple:
 - ❑ Access control rules specified in IdP/DC
 - ❑ Access control rules enforced by SP/S

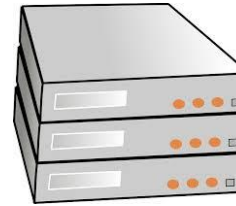
OAuth: Functionality



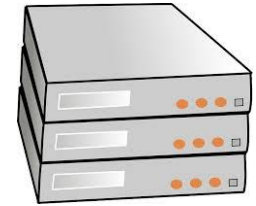
Scenario (I)

- ❑ **Different** administrative domains
- ❑ Communication across the **Internet**
- ❑ Every interaction on **HTTPS**
- ❑ **User** has credentials on **both** webapps
- ❑ Managed independently of each other

WebApp 2



WebApp 1



Browser

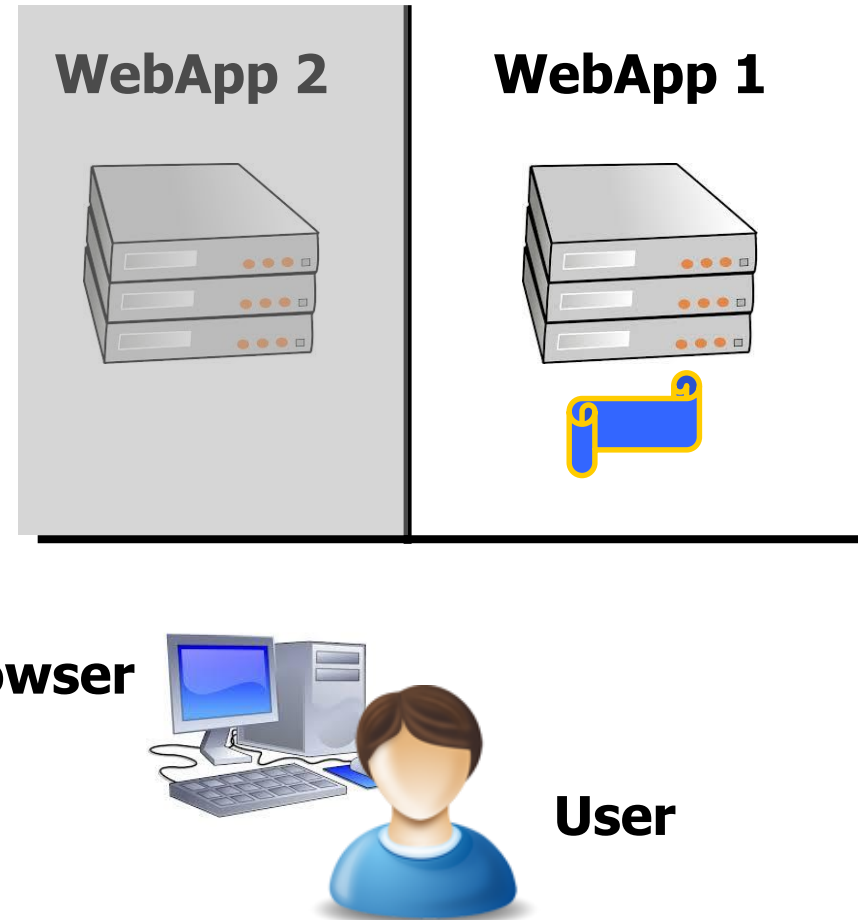


User



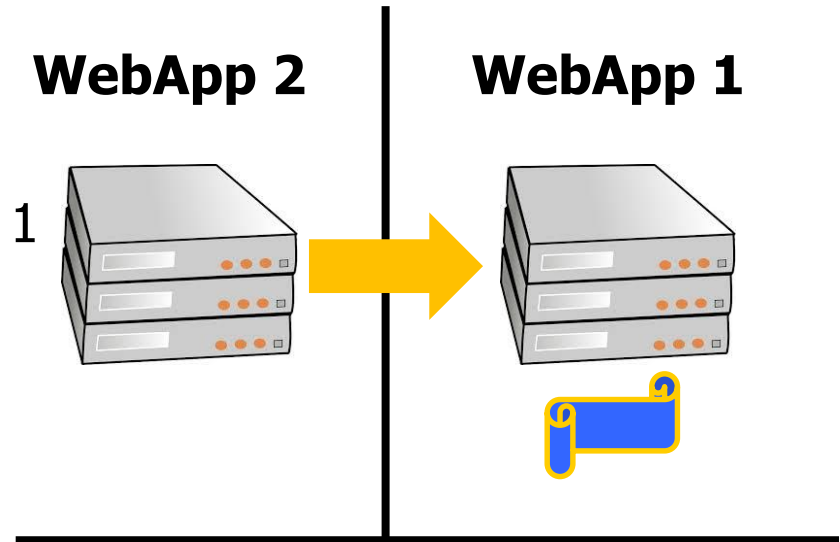
Scenario (II)

- ❑ **User** owns **Resources** on WebApp 1
 - ❑ Google: Drive, Gmail, Calendar...
 - ❑ Facebook: Wall, Friend list...
 - ❑ Twitter: tweets,...
 - ❑ ...



Objective

- ❑ User **delegates** WebApp 2 to operate on his resources on WebApp 1
 - ❑ **Directly** (e.g., when User is **not** logged in)
 - ❑ For **long** time periods, or until **revocation**



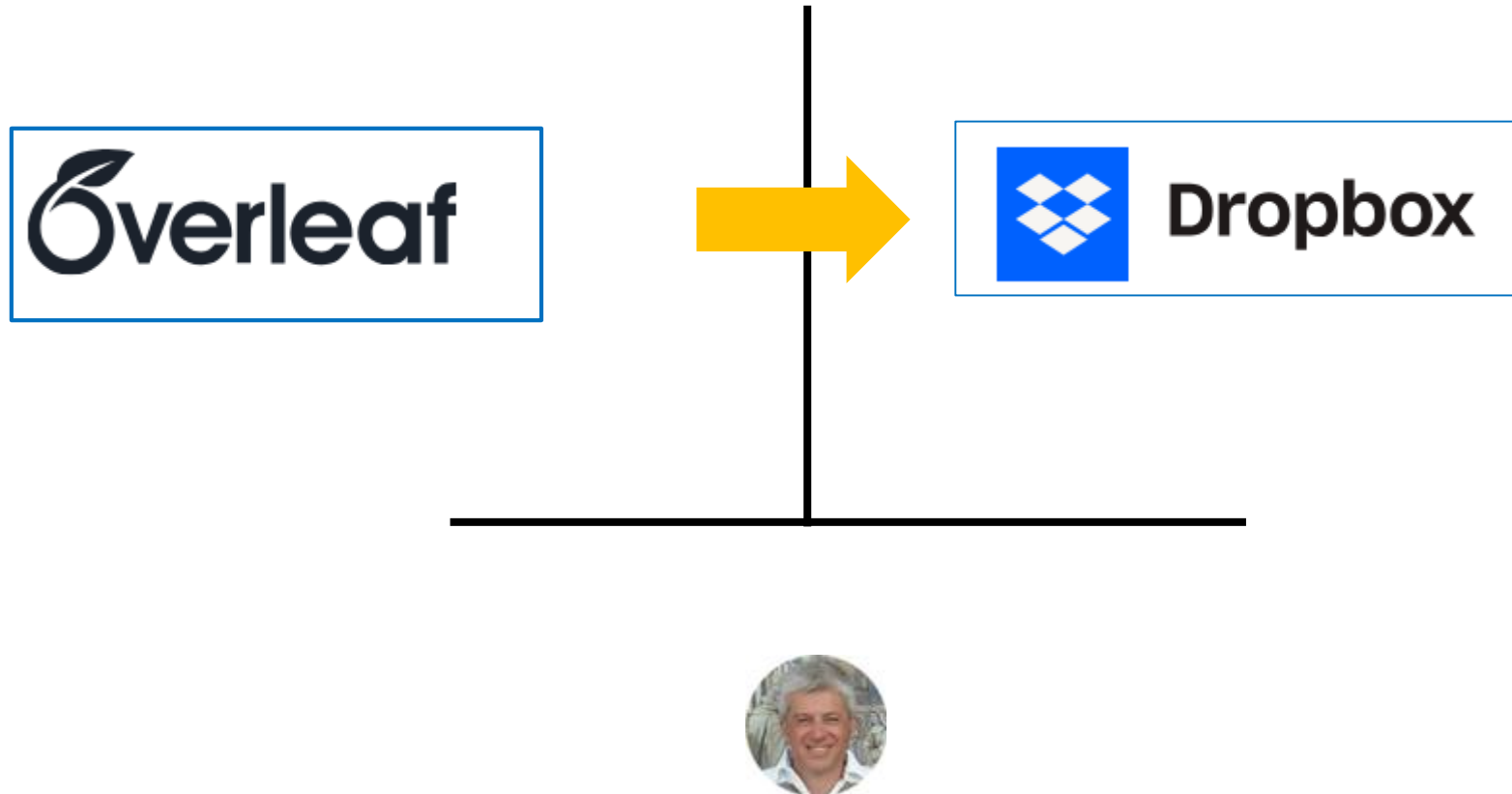
- ❑ WebApp 2 does **not** know User credentials on WebApp 1

Browser



User

Example (I-a)



Example (I-b)



Account settings

Project synchronisation



Dropbox Sync

Keep your Overleaf projects in sync with your Dropbox account. Changes in Overleaf are automatically sent to your Dropbox account, and the other way around. [Learn more about Dropbox Sync](#)

Link



GitHub Sync

With GitHub Sync you can link your Overleaf projects to GitHub repositories, create new commits from Overleaf, and merge commits from GitHub. [Learn more about GitHub Sync](#)

Link

Example (I-c)



Dropbox



Overleaf would like access to its own folder,
Apps › **Overleaf**, inside your Dropbox. [Learn more](#)

Cancel

Allow

Example (I-d)



Dropbox

View Dropbox content

View and make changes to your Dropbox content while using these apps outside of Dropbox. If you don't recognize an app, disconnect it.



Rakuten Kobo
Rakuten Kobo Inc.



Dropbox
Dropbox

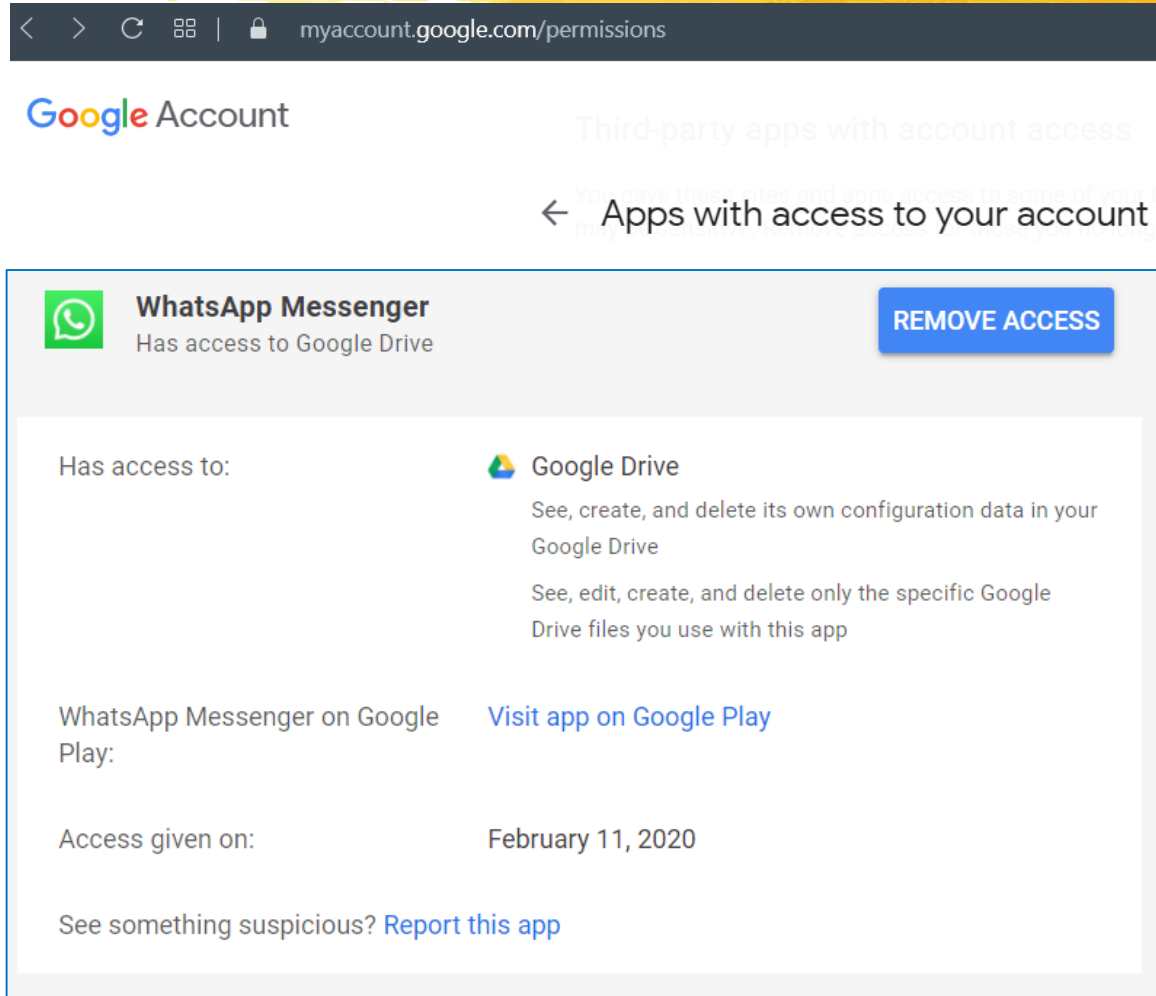


Joplin
Laurent Cozic



ShareLaTeX
ShareLaTeX

Example (II)




The screenshot shows a web browser at the URL `myaccount.google.com/permissions`. The page title is "Google Account". Below the title, there is a section titled "Third-party apps with account access". A back arrow and the text "Apps with access to your account" are visible. The main content area shows a card for "WhatsApp Messenger" with the subtitle "Has access to Google Drive". A blue button labeled "REMOVE ACCESS" is in the top right corner of the card. Below the app name, it says "Has access to:" followed by "Google Drive". The permissions listed are: "See, create, and delete its own configuration data in your Google Drive" and "See, edit, create, and delete only the specific Google Drive files you use with this app". There is a link to "Visit app on Google Play". The "Access given on:" date is "February 11, 2020". At the bottom, there is a link to "Report this app" with the text "See something suspicious?".

myaccount.google.com/permissions


Google Account

Third-party apps with account access

← Apps with access to your account

 **WhatsApp Messenger**
Has access to Google Drive [REMOVE ACCESS](#)

Has access to:

 **Google Drive**
See, create, and delete its own configuration data in your Google Drive
See, edit, create, and delete only the specific Google Drive files you use with this app

WhatsApp Messenger on Google Play: [Visit app on Google Play](#)

Access given on: February 11, 2020

See something suspicious? [Report this app](#)

My advice

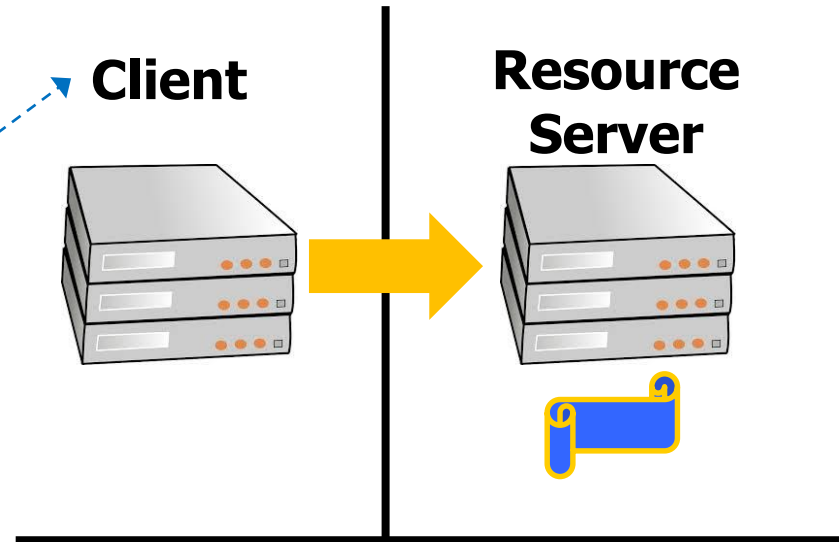


Every now and then:

- ❑ Foreach of your "important services"
(Google / Facebook / Instagram / ...)
 1. Look at the **access rights** you have granted to **third parties**
(connected apps / integrations / ...)
 2. Are they really useful to you?
 3. IF not really useful THEN revoke

OAuth Terminology

Confusing name:
for resource owner it is a **server**



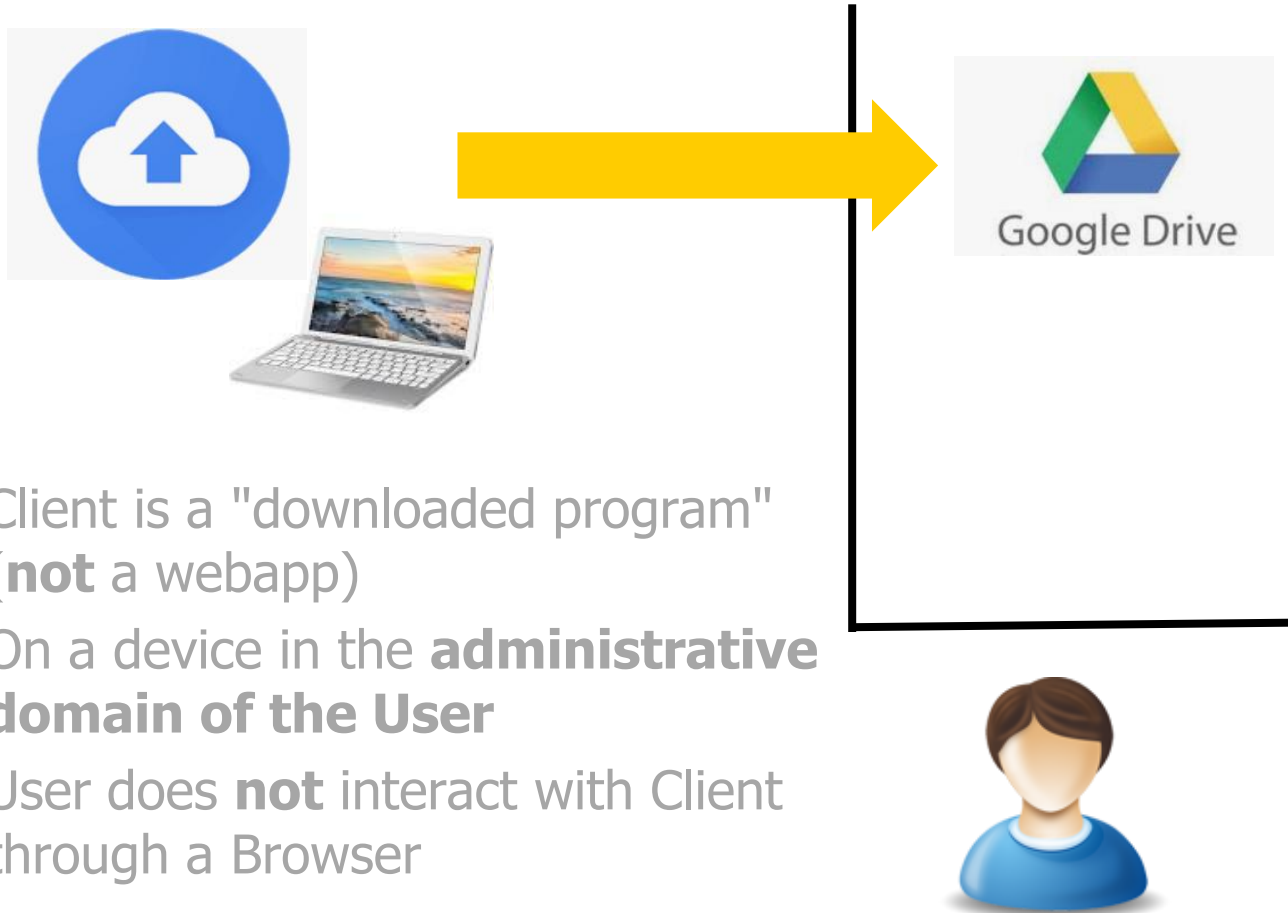
Many other use cases (I-a)



- ❑ Client is a "downloaded program" (**not** a webapp)
- ❑ On a device in the **administrative domain of the User**
- ❑ User does **not** interact with Client through a Browser

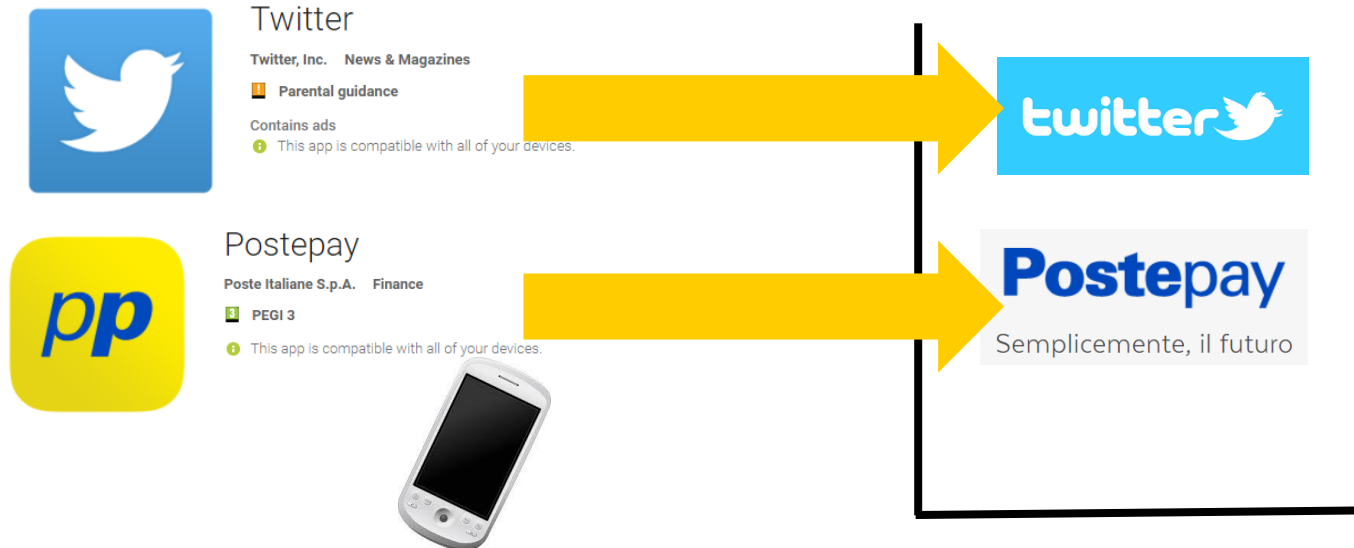


Many other use cases (I-b)



- ❑ Client is a "downloaded program" (**not** a webapp)
- ❑ On a device in the **administrative domain of the User**
- ❑ User does **not** interact with Client through a Browser

Many other use cases (II)



- ❑ Client is a "downloaded program" (**not** a webapp)
- ❑ On a device in the **administrative domain of the User**
- ❑ User does **not** interact with Client through a Browser



Many other use cases (III)



- ❑ Client is a "locally developed program"
- ❑ On a device in the **administrative domain of the User**
- ❑ User does **not** interact with Client through a Browser



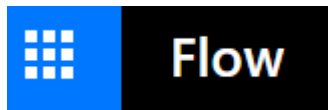
GitHub = OAuth Resource Server



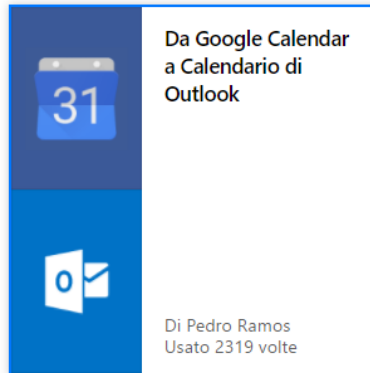
- ❑ My apps/Projects
 - ❑ Delegated to operate on **my** GitHub repositories



"Cloud Workflows"



Now called
Power Automate



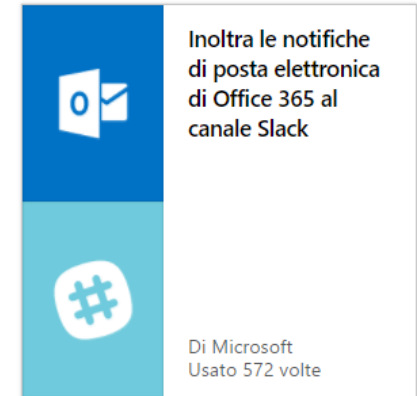
Da Google Calendar a Calendario di Outlook

Di Pedro Ramos
Usato 2319 volte



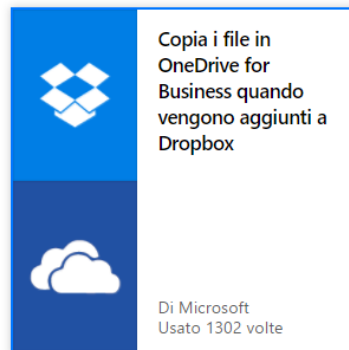
Ricevi le attività Wunderlist quotidiane nella posta elettronica

Di Anoj Pillai
Usato 535 volte



Inoltra le notifiche di posta elettronica di Office 365 al canale Slack

Di Microsoft
Usato 572 volte



Copia i file in OneDrive for Business quando vengono aggiunti a Dropbox

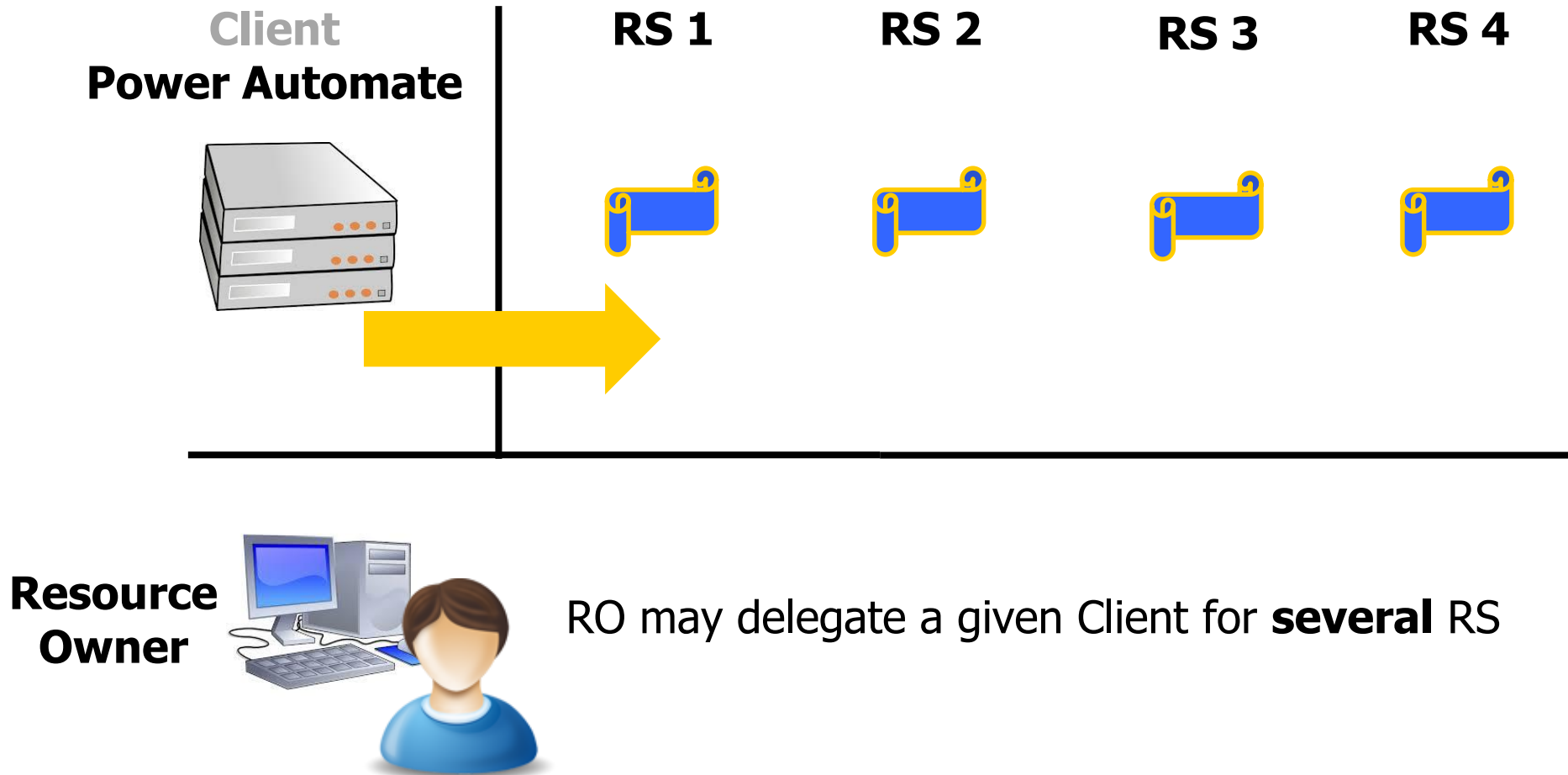
Di Microsoft
Usato 1302 volte



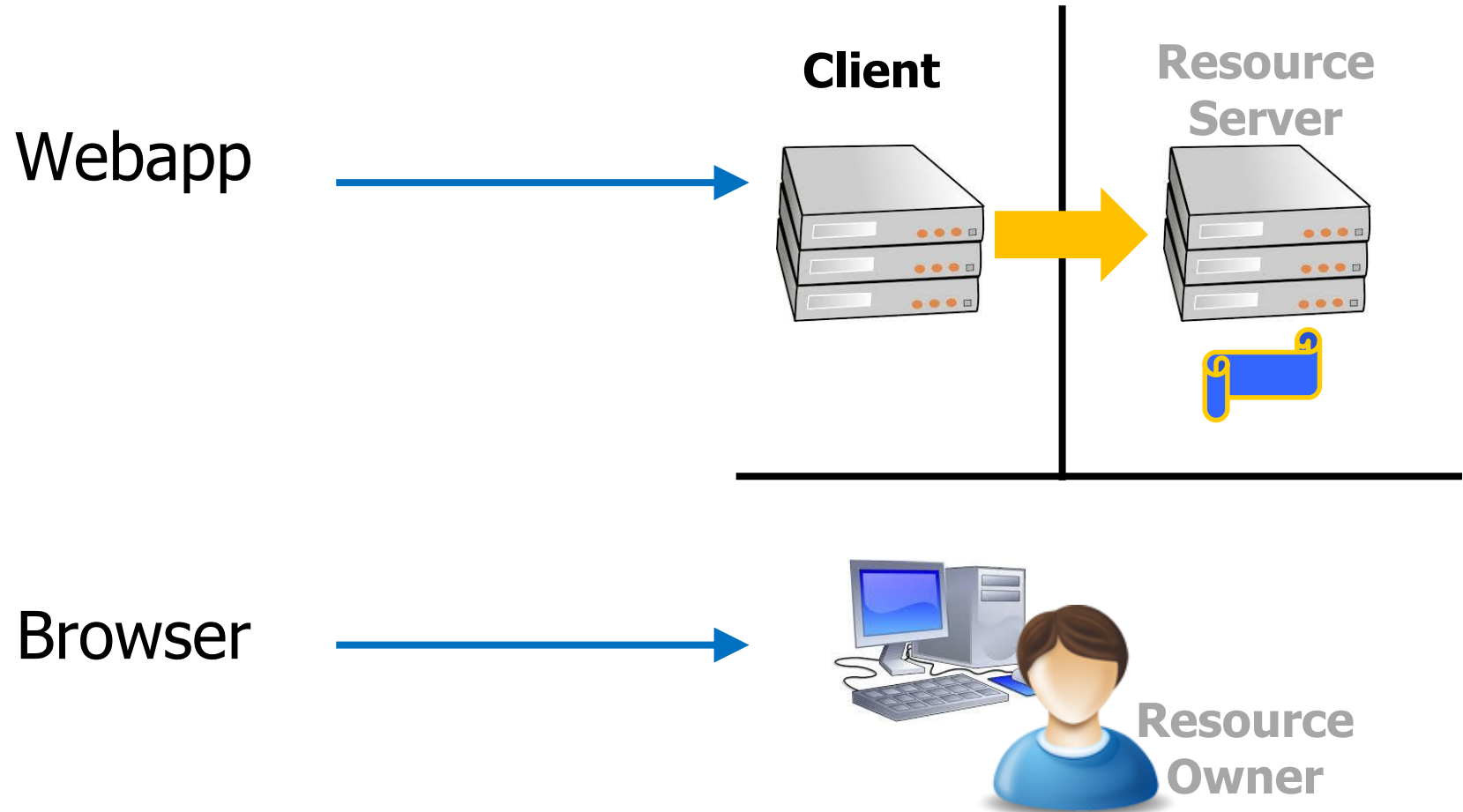
Salva i tweet che includono un hashtag specifico in un elenco SharePoint

Di Microsoft
Usato 672 volte

...yet another use case



For now



This is NOT OAuth (I)

Log in to Overleaf

Email

Password

Log in with your email

or

Log in with IEEE

Log in with Google

Log in with Twitter

Log in with ORCID

Email or username

Password


☒ Stay logged in [Forgot password?](#)

Log in

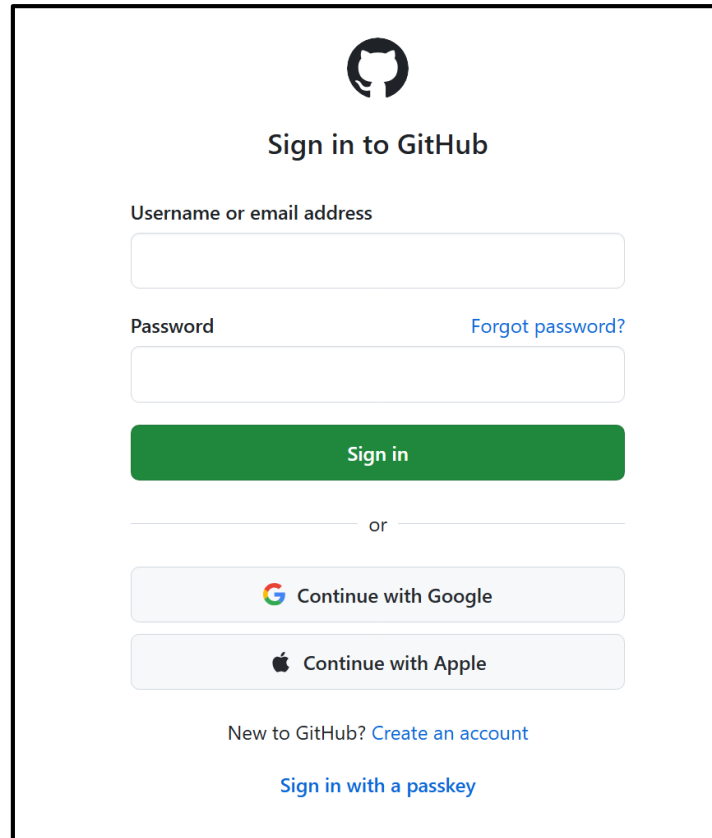
OR

Log in with Facebook

Log in with Google



This is NOT OAuth (II)



The image shows the GitHub sign-in page. At the top is the GitHub logo (Octocat). Below it is the text "Sign in to GitHub". There are two input fields: "Username or email address" and "Password". To the right of the password field is a link "Forgot password?". Below the input fields is a green "Sign in" button. Below the button is a horizontal line with the word "or" in the center. Below the line are two buttons: "Continue with Google" (with the Google logo) and "Continue with Apple" (with the Apple logo). At the bottom, there is a link "New to GitHub? Create an account" and a link "Sign in with a passkey".


Sign in to GitHub


Username or email address

Password [Forgot password?](#)

Sign in

or

 Continue with Google

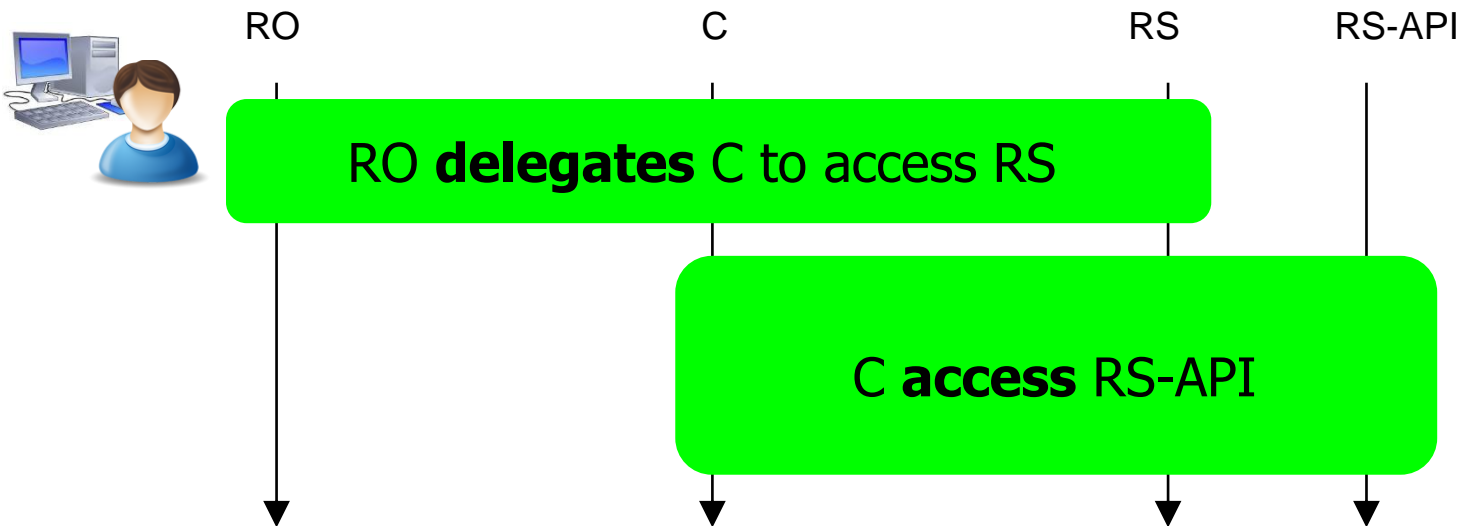
 Continue with Apple

New to GitHub? [Create an account](#)

[Sign in with a passkey](#)

OAuth Functionality in a nutshell

- ❑ **Open** standard designed to work with **HTTPS**
- ❑ Specifies a process for **resource owners** to **delegate third-party client access** to their **server resources**
- ❑ ...**without** sharing their **credentials**



Resources and Access Rights (I)



- ❑ **Each Resource Server** defines:
 - ❑ What a **resource** is
 - ❑ Which **operations** can be done on resources
 - ❑ How **access rights** are described (**granularity** of delegation)
 - ❑ Operations X, Y on all files
 - ❑ Operations X, Y only on files created by this client
 - ❑ Operation Z on all the emails
 - ❑ ...

Resources and Access Rights (II)

- ❑ What a **resource** is
- ❑ Which **operations** can be done on resources
- ❑ How **access rights** are described (**granularity** of delegation)



- ❑ **Scope** = string
- ❑ OAuth defines:
 - ❑ Specifies how to transfer scope information
 - ❑ Does **not** specify its **meaning**

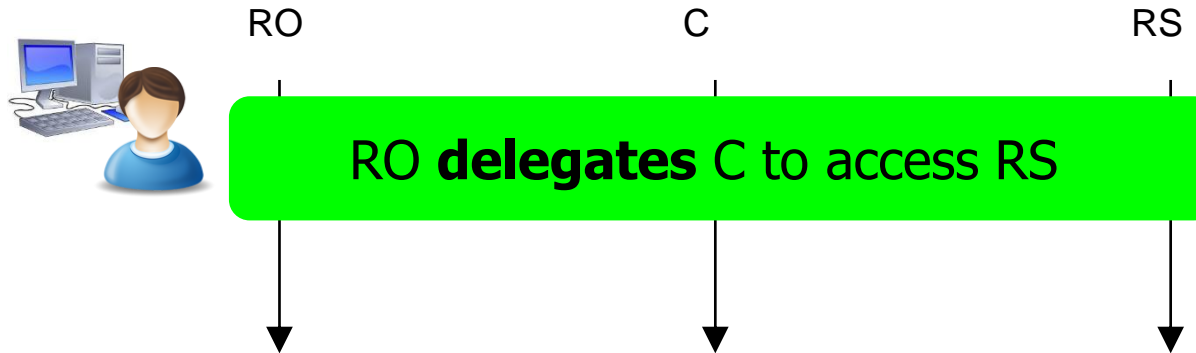
It is a **KEY** technology



- ❑ Supported by all large players
 - ❑ Google, Facebook, Twitter, Microsoft, LinkedIn
 - ❑ Paypal
 - ❑ Amazon, Dropbox, Evernote, Instagram, Yahoo!
 - ❑ ...

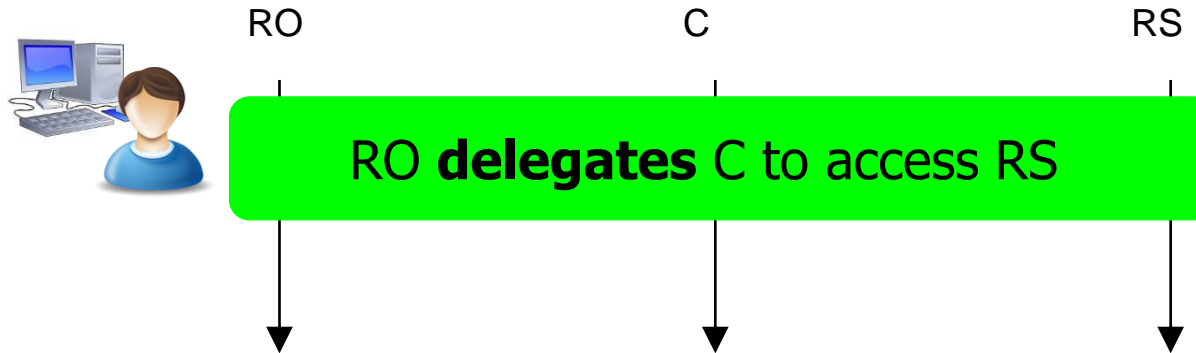
https://en.wikipedia.org/wiki/List_of_OAuth_providers

Delegation phase (I-a)



- ❑ **Many** flavours
- ❑ **Always** start at the **client**
 - ❑ RO **authenticates** on both C and RS (obviously)
- ❑ Try one yourself
 - ❑ Allow Overleaf to access Dropbox
 - ❑ Connected apps / Integrations / ...

Delegation phase (II)

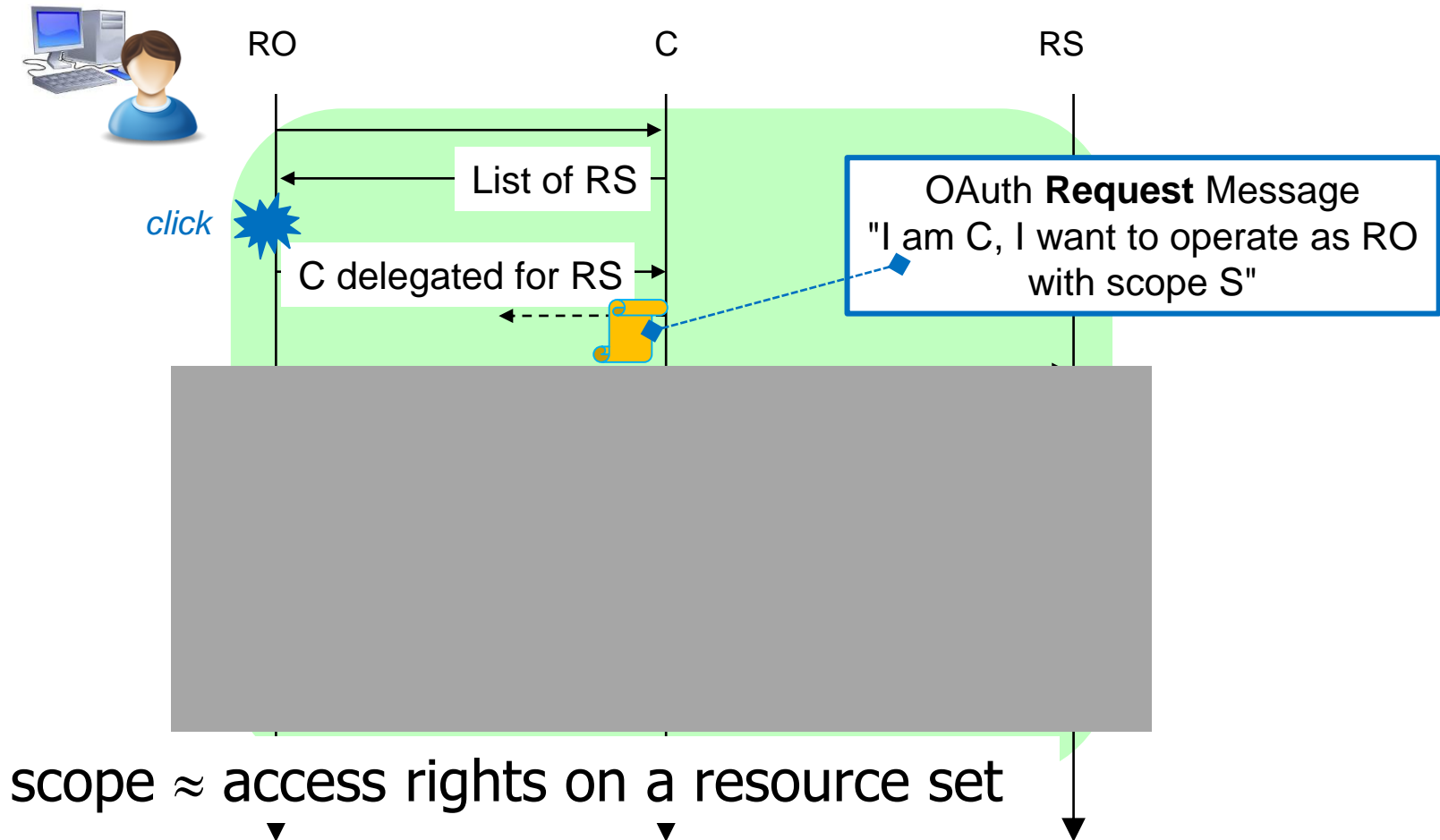


- ❑ Many flavours

- ❑ Assumption:

- ❑ RO is already **authenticated** on both C and RS
 - ❑ If not, obvious additions

Delegation phase (III-a)



Delegation Example:

Client: "List of RS"



Account settings

Project synchronisation



Dropbox Sync

Keep your Overleaf projects in sync with your Dropbox account. Changes in Overleaf are automatically sent to your Dropbox account, and the other way around. [Learn more about Dropbox Sync](#)

Link



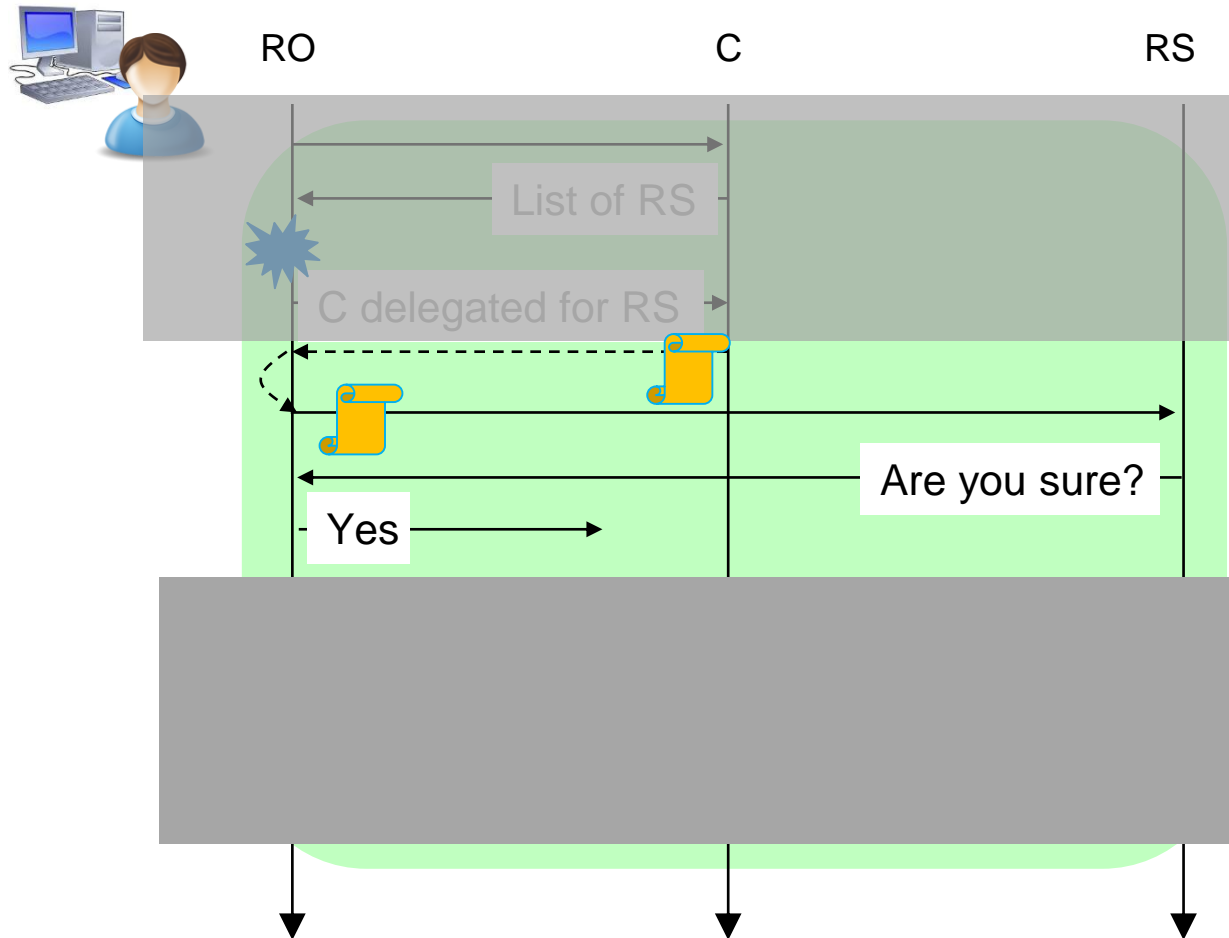
GitHub Sync

With GitHub Sync you can link your Overleaf projects to GitHub repositories, create new commits from Overleaf, and merge commits from GitHub. [Learn more about GitHub Sync](#)

Link

Page in **authenticated** session at the **Client**

Delegation phase (III-b)



Delegation Example 1:

RS: "Are you sure?"



Dropbox

Client



Scope

Overleaf would like access to its own folder,
Apps › **Overleaf**, inside your Dropbox. [Learn more](#)

Cancel

Allow

Delegation Example 2:

RS: "Are you sure?"

Automated generation
of forms from slides



wants additional
access to your Google
Account


 bartoli.alberto@gmail.com

Select what
Automated generation of forms from slides can
access

☐ Select all

Scope

-  See information about your Google Drive files. ☐
[Learn more](#)
-  See, edit, create and delete all your Google Sheets
spreadsheets. [Learn more](#) ☐



 **Automated generation of forms from slides already
has some access**

See the [2 services](#) to which Automated generation of forms from
slides has some access.

Client

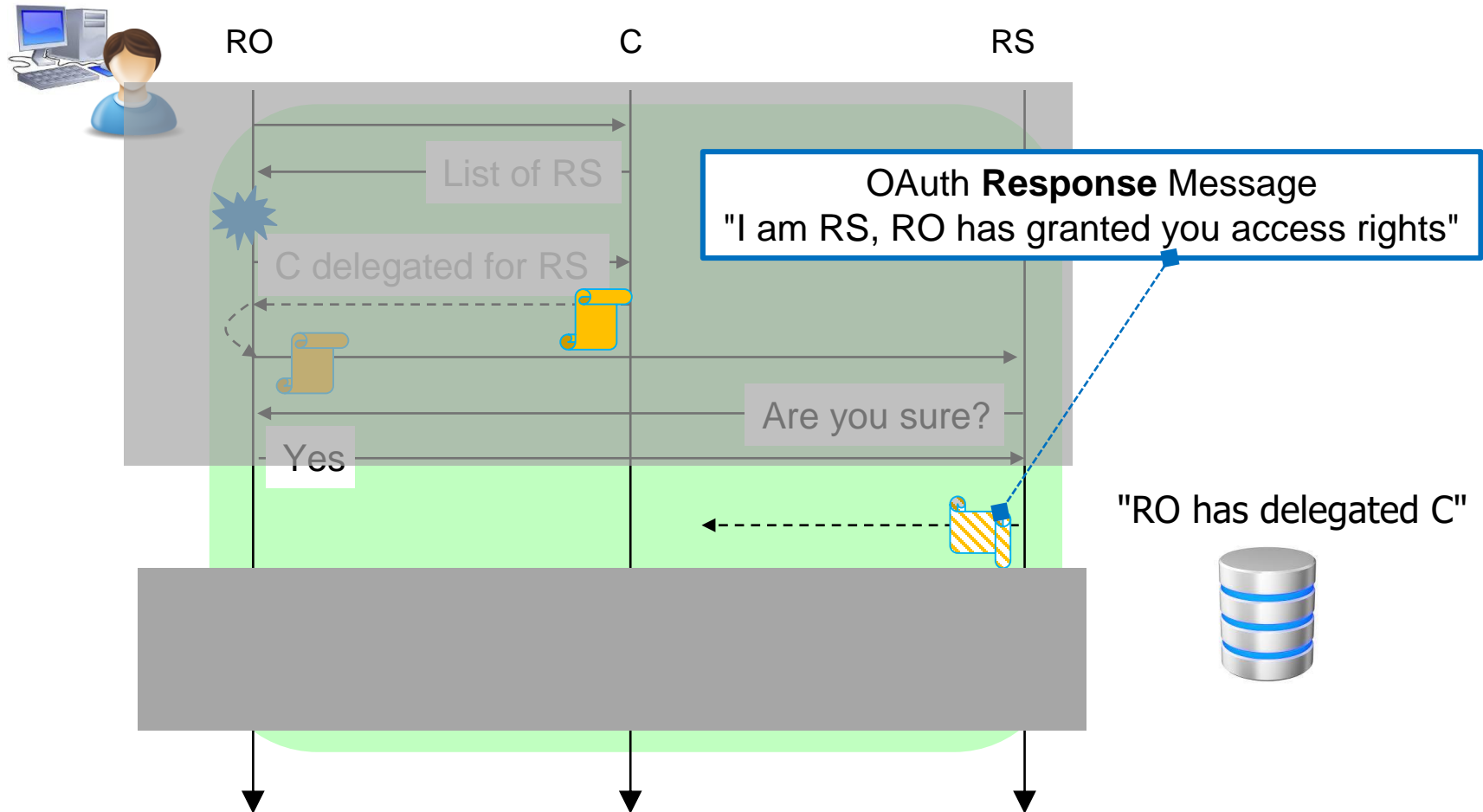
Automated generation of forms from slides has
this access

You can always remove any access in your [Google Account](#).

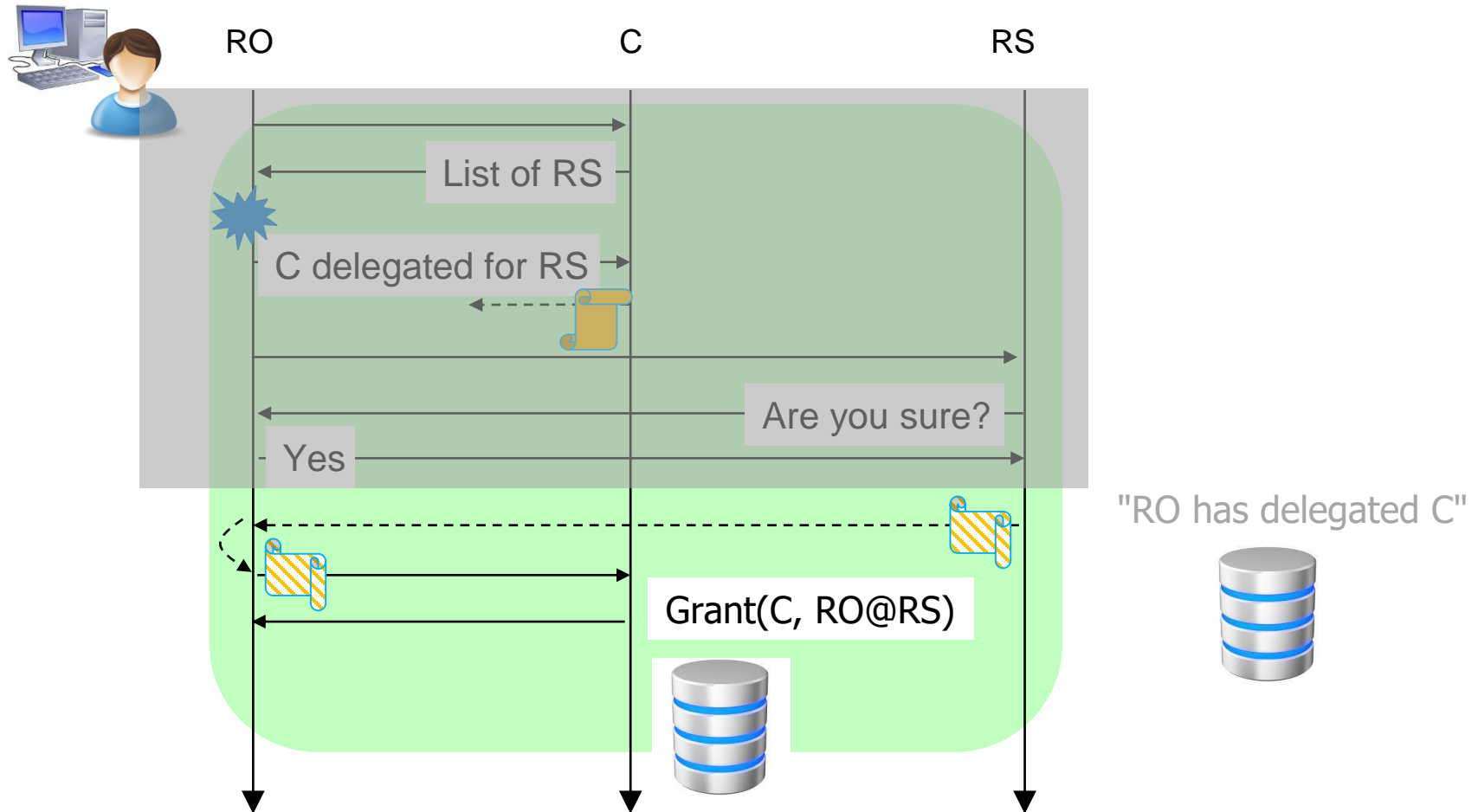
-  See all responses to your Google Forms forms ☐
-  See, edit, create and delete all your Google
Forms ☐

Done

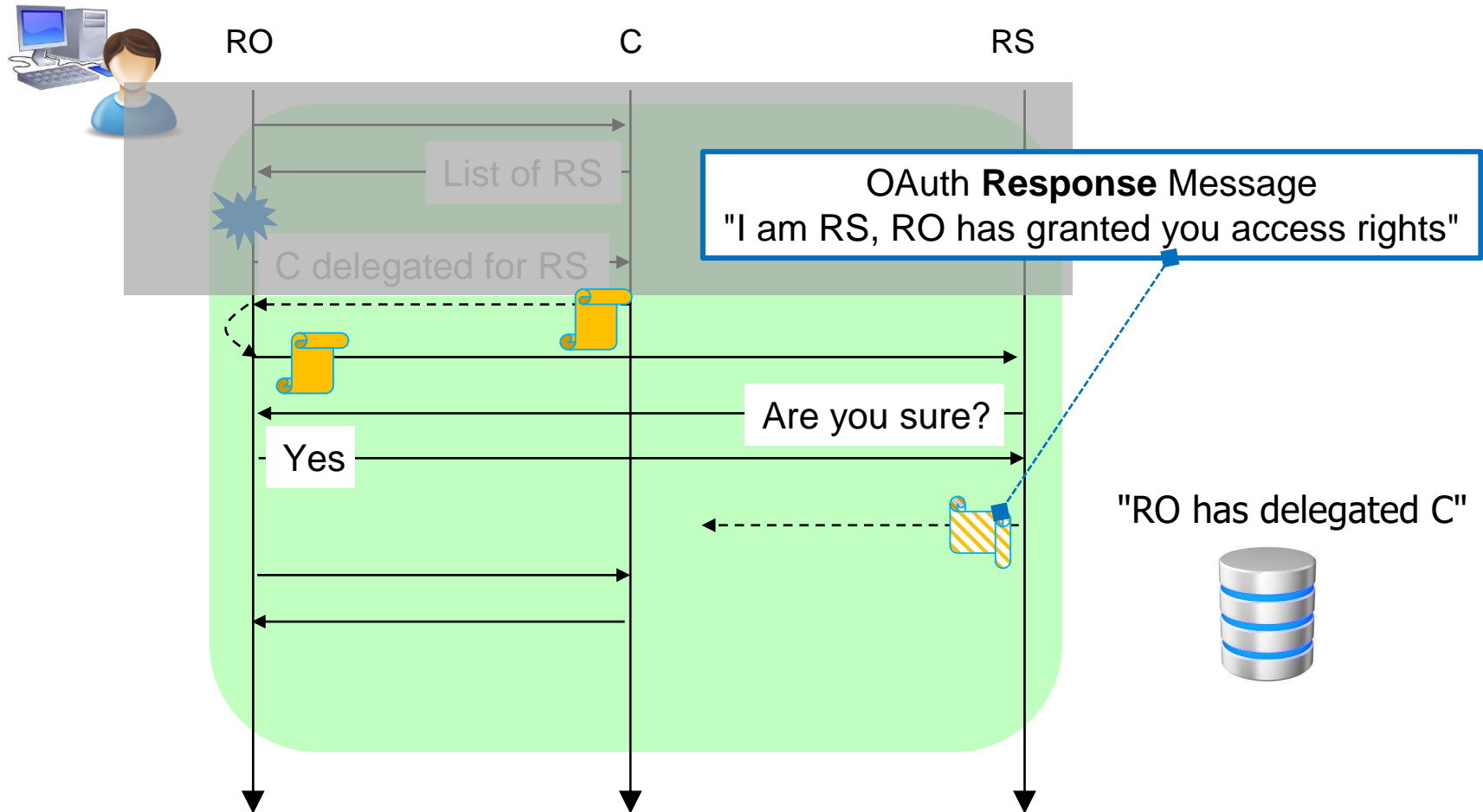
Delegation phase (III-c)



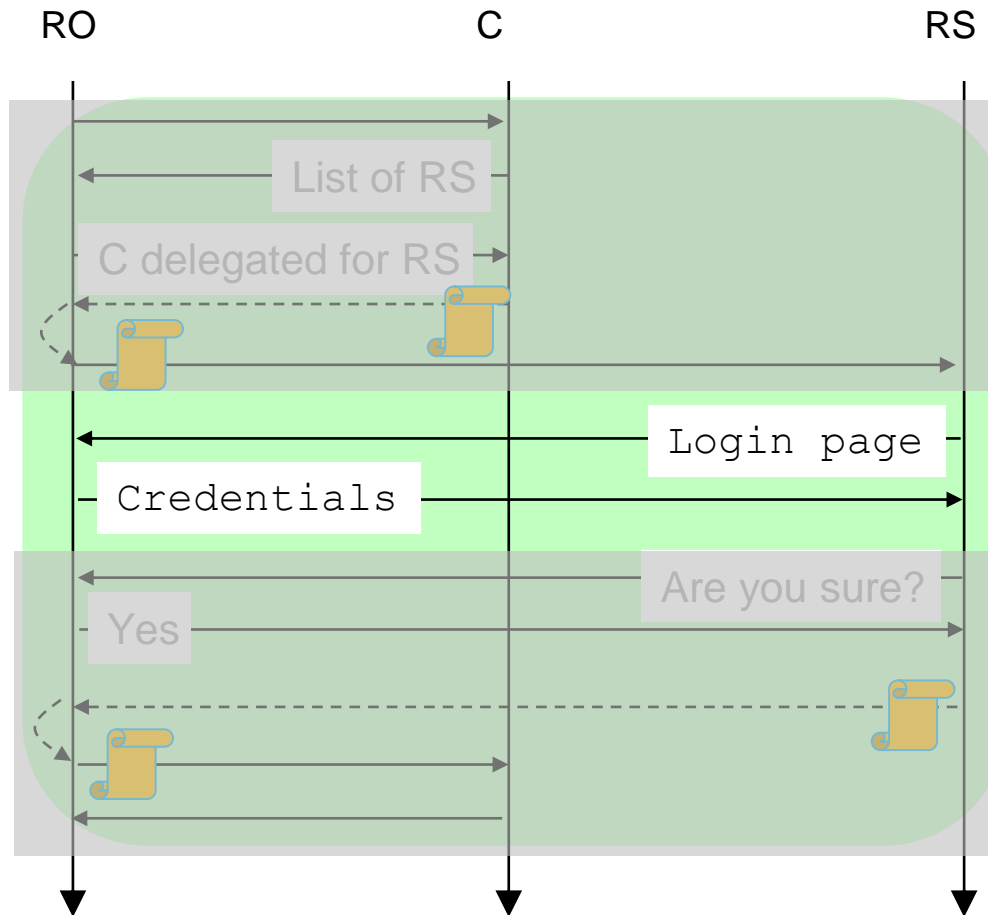
Delegation phase (III-d)



Delegation (more details later) (III)



Remark



if RO not authenticated

What OAuth specifies (I)



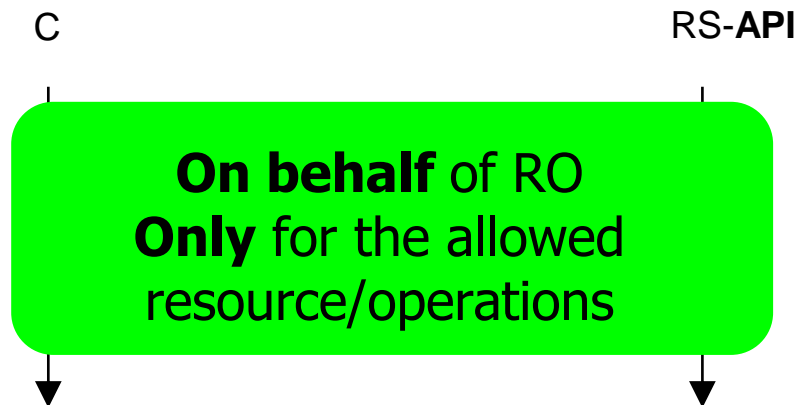
- ❑ How to **represent** authorization information
- ❑ How to **enclose** it in HTTP messages
- ❑ How to **exchange** it between browser, C, RS
- ❑ The **meaning** of that information is RS-specific
- ❑ Each Resource Server defines:
 - ❑ What a **resource** is
 - ❑ Which **operations** can be done on resources
 - ❑ How **access rights** are described (granularity of delegation)

What OAuth specifies (II)

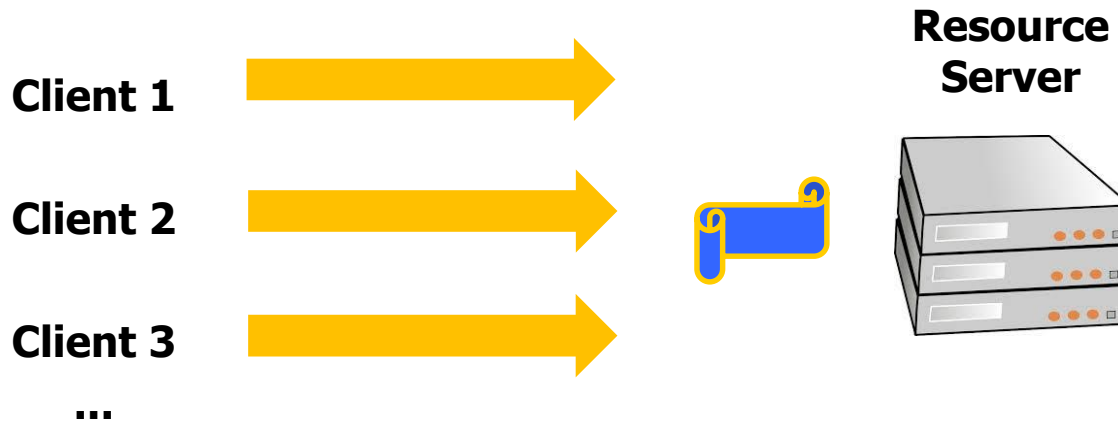
- How to **exchange** it between browser, C, RS
- Browser **flow**
 - Message exchange pattern cannot be arbitrary:
it must follow the behavior of the browser
(req/resp, redirections, ...)
- **Many other** OAuth flows
 - Generic program, C, RS
 - Javascript loaded by the browser, C, RS
 - ...

Access

- How to **represent** authorization information
- How to **enclose** it in HTTP messages
- Resource Access occurs with an **RS-specific** API



Great! (I)



- ❑ **Any kind** of program
- ❑ ...running **anywhere**

- ❑ **Centralized** control of **all** delegations
- ❑ Can **revoke** at any time with just **one click**



Great! (II)



- ❑ Typical approach **before** OAuth
 1. RO obtains API key from RS
 2. RO shares API key with Client
 3. Client embeds API key in API requests

- ❑ No **uniform** framework: every RS had its own framework

- ❑ No **automatic** flow
- ❑ **Revocation** of API keys much more complex
- ❑ **Restricting access rights** for a given API key much more complex (or impossible)

OAuth: Implementation



Preamble



- ❑ OAuth:
 - ❑ Composed of a **lot** of **variants** (really a lot)
 - ❑ Message-exchange patterns ("**flows**")
 - ❑ Message and token contents
 - ❑ Includes a **lot** of **details** (really a lot)
- ❑ I admit I do not know/remember many of them
- ❑ We will focus on a few of the most important scenario

OAuth2 Threat Model



- ❑ RFC 6819
OAuth2.0 Threat Model and Security Considerations
(65 pg)
- ❑ In a nutshell
 - ❑ Threat Model **Network Attacker**
 - ❑ Cannot **alter any code / steal any information**
on RO, C, RS

HTTPS

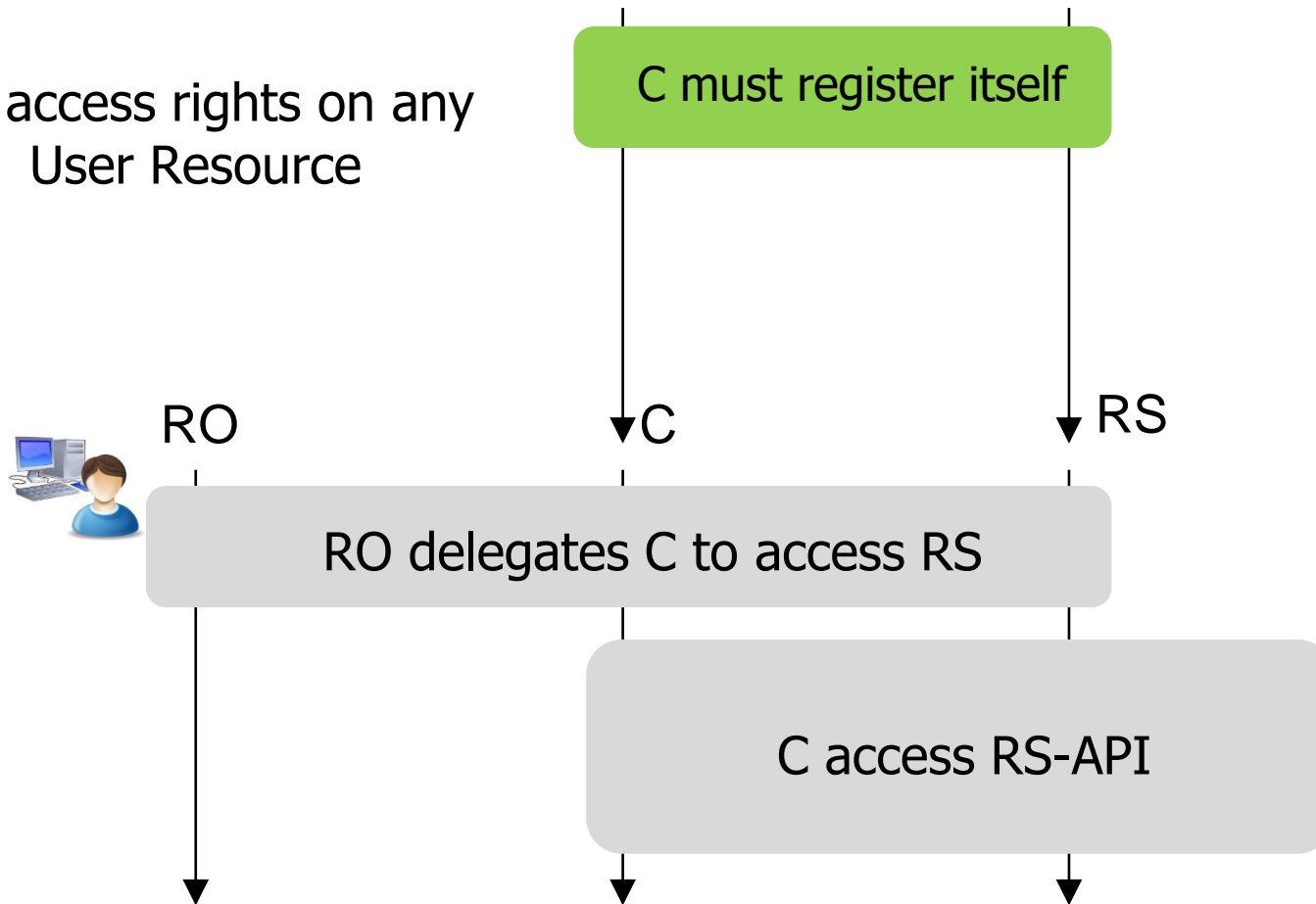


- ❑ Every communication on **HTTPS**
 - ❑ Server authentication
 - ❑ Secrecy
 - ❑ Integrity

- ❑ **First approximation:** Attacker can only DoS

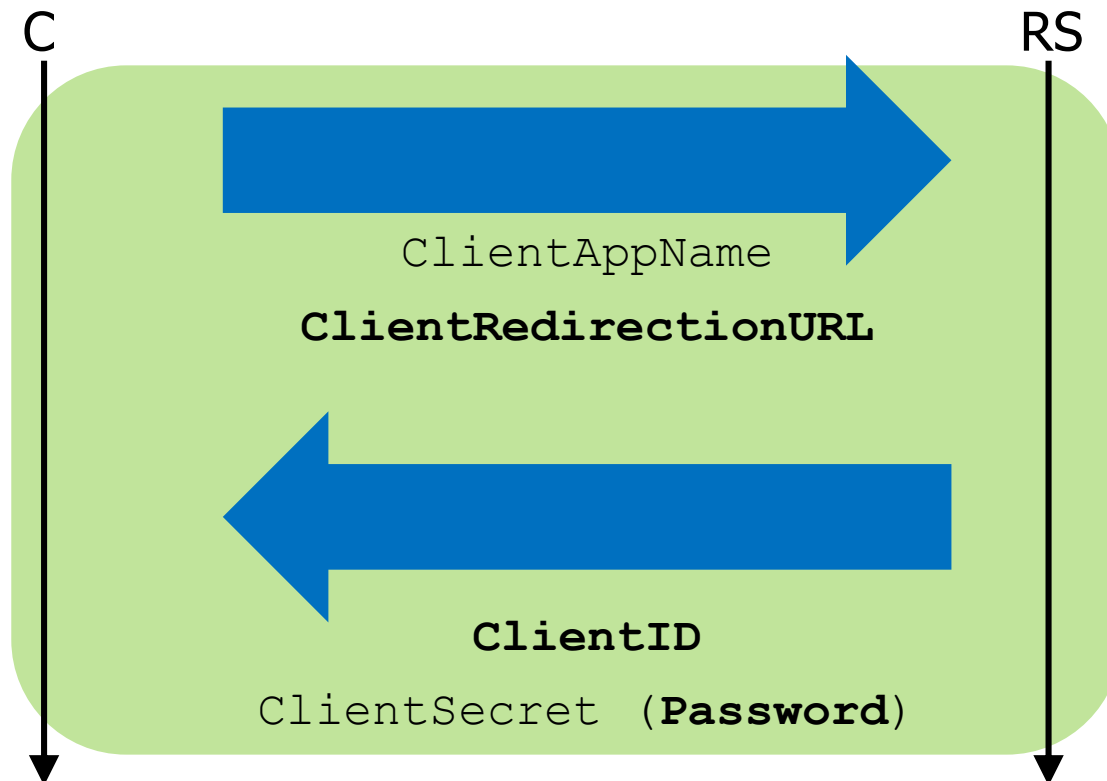
Phase 0 (I)

No access rights on any
User Resource

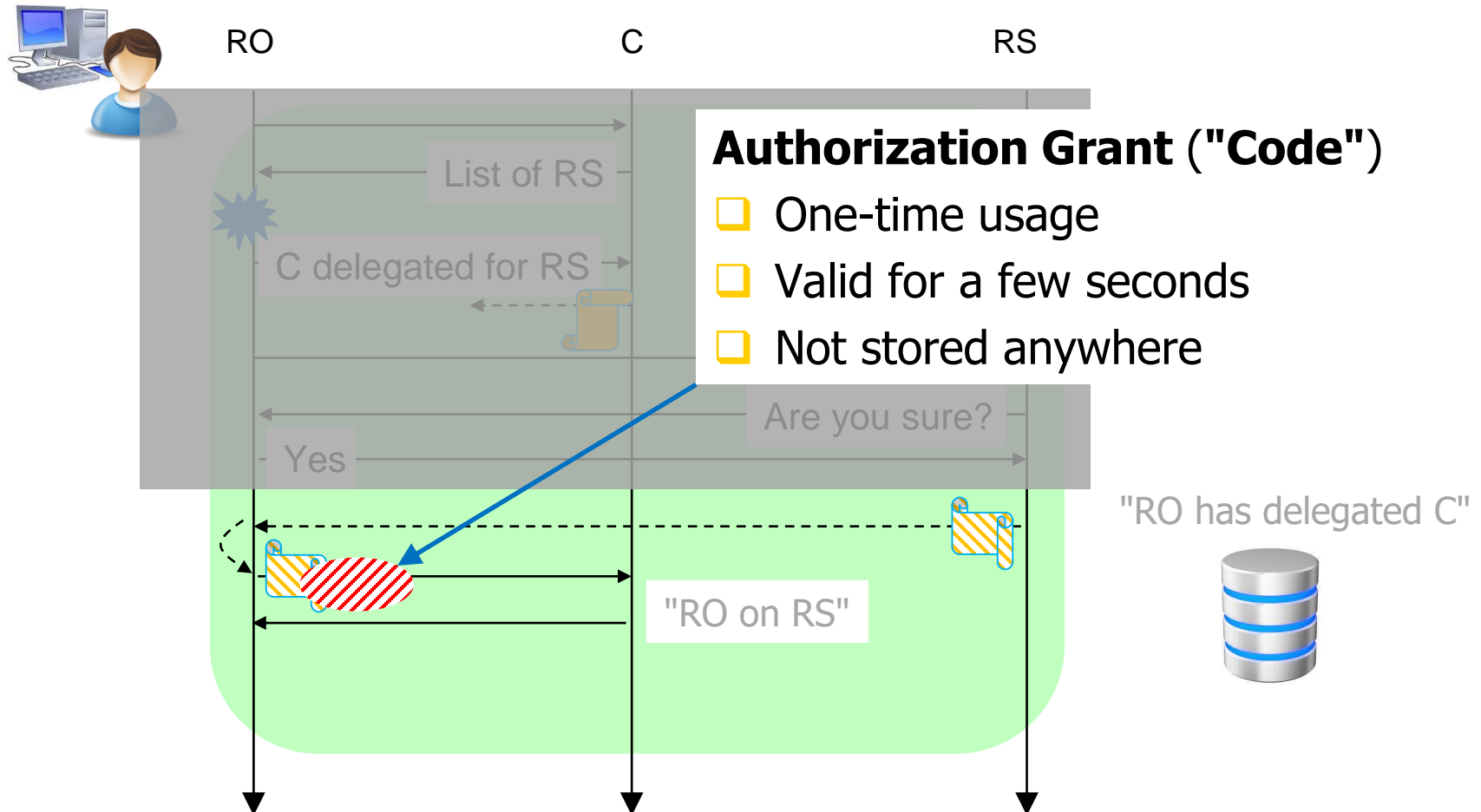


Phase 0 (II)

Non-standard procedure specified by each RS



Delegation phase (I)

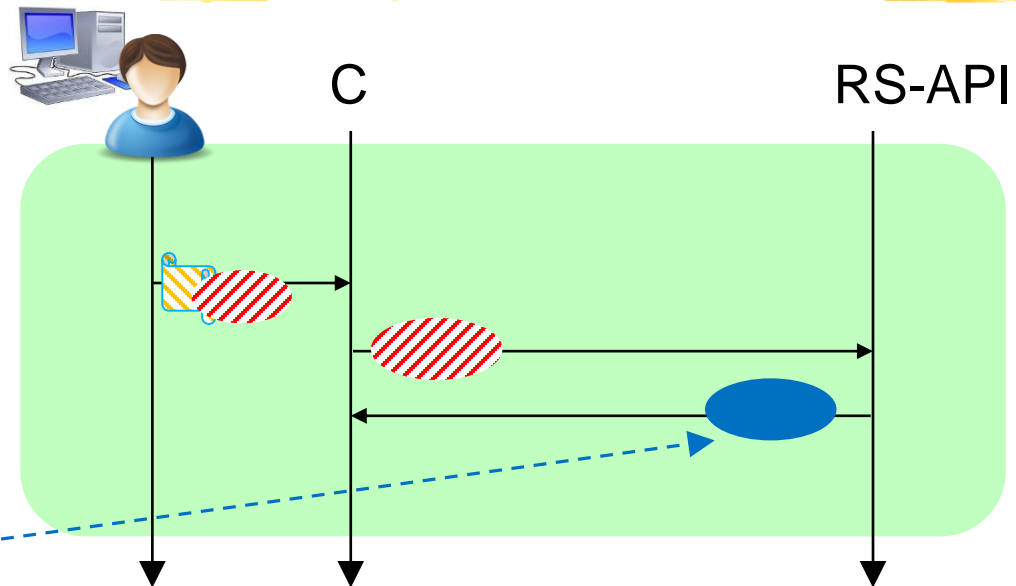


Delegation phase (II)

- ❑ Authorization Code
 - ❑ One-time usage
 - ❑ Valid for a few seconds
 - ❑ Not stored anywhere

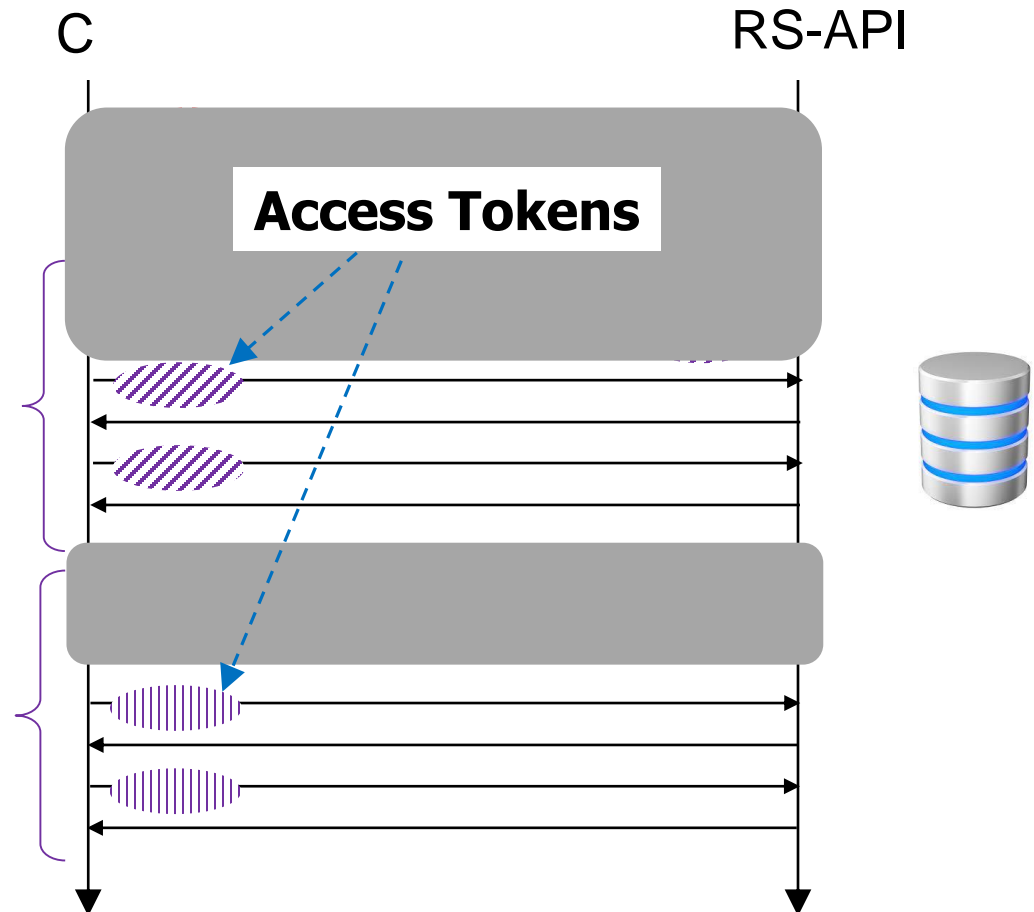


- ❑ **Refresh Token**
 - ❑ Months / Revocation
 - ❑ **Stored in C database**



Access Phase (I)

- ❑ RS-API expects to receive an **Access Token**
- ❑ Encodes Resources and Access Rights
- ❑ Lifetime few minutes



Access Phase (II)

- Authorization Grant (Code)

- Very few minutes



- Refresh Token

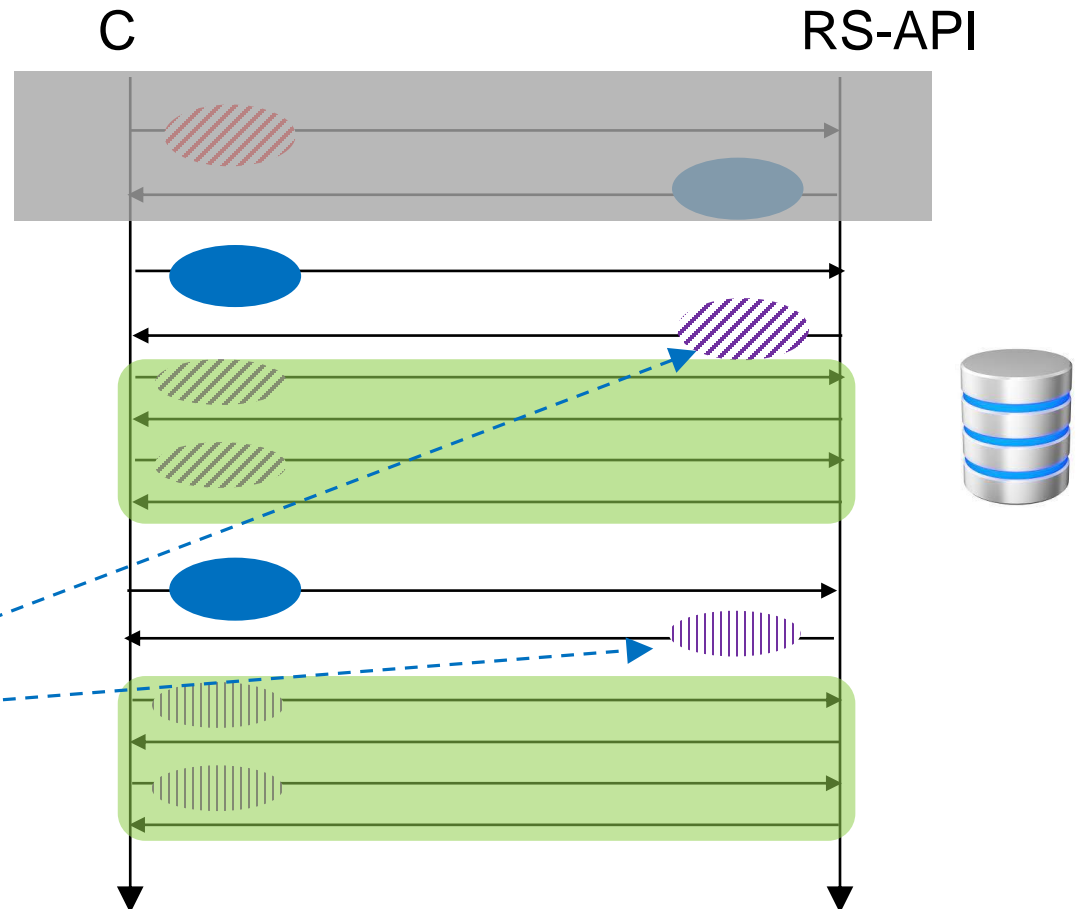
- Months / Revocation

- Stored in C database

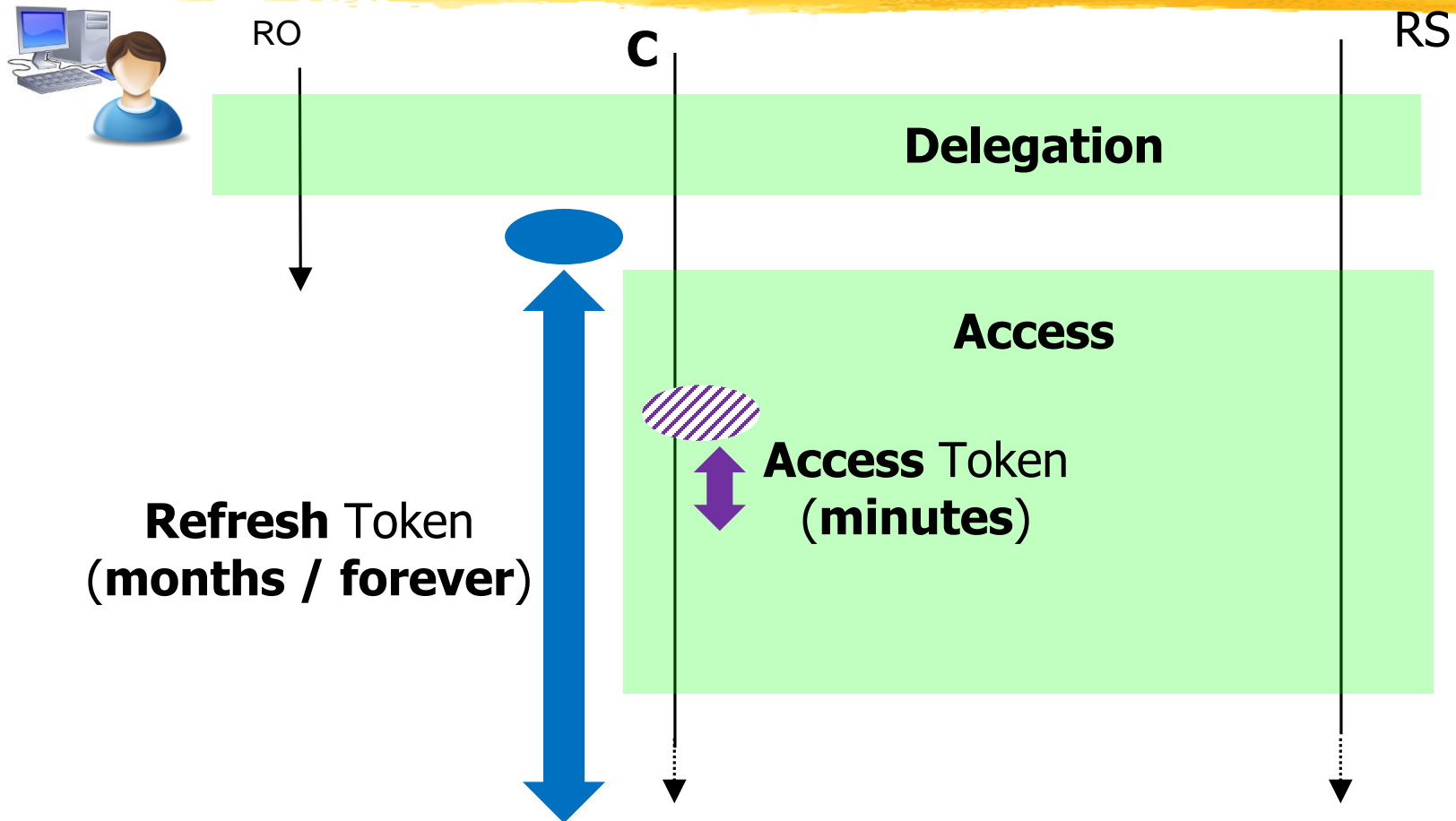


- Access Token

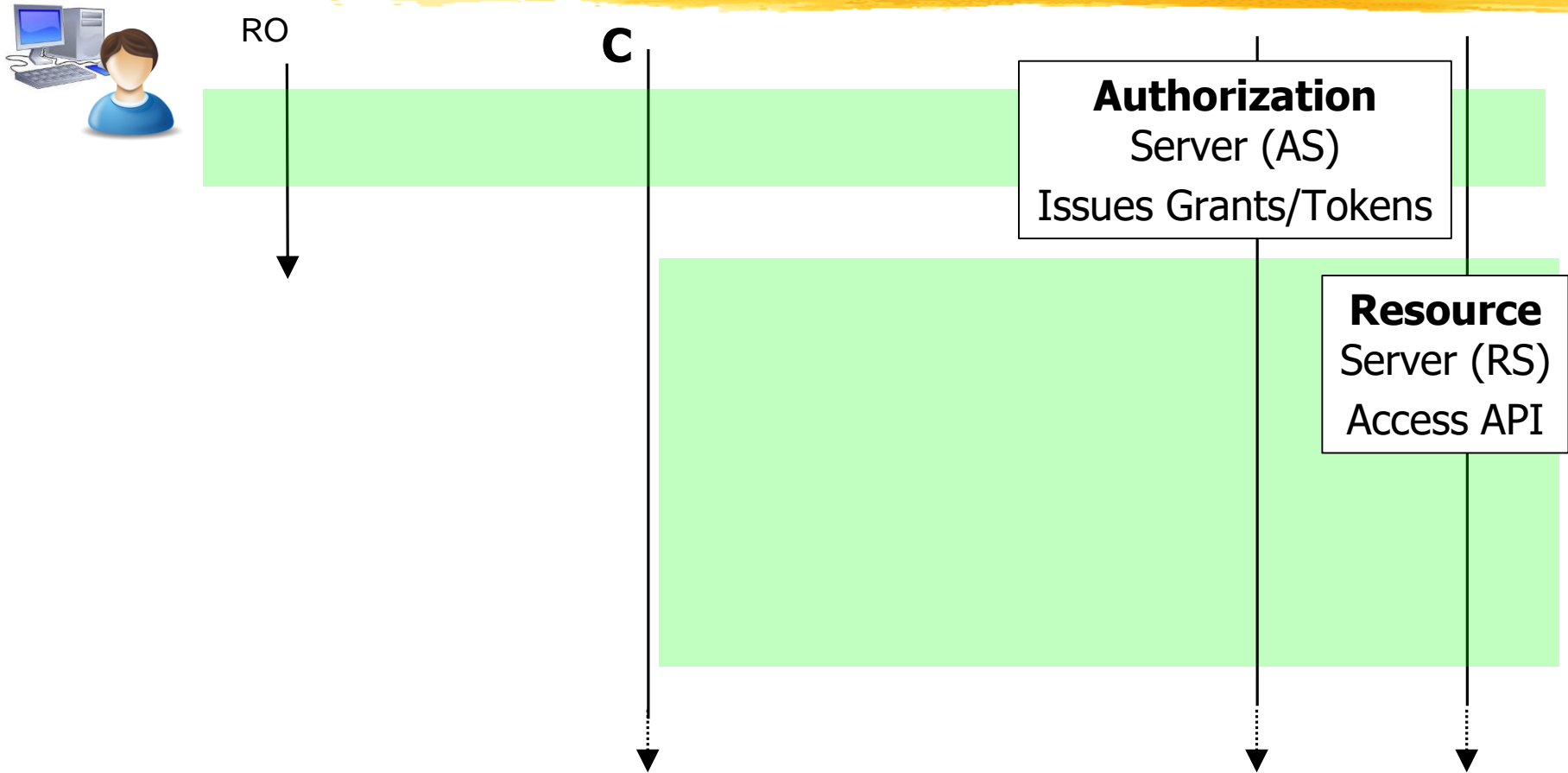
- Few minutes



Access Phase: Timeline



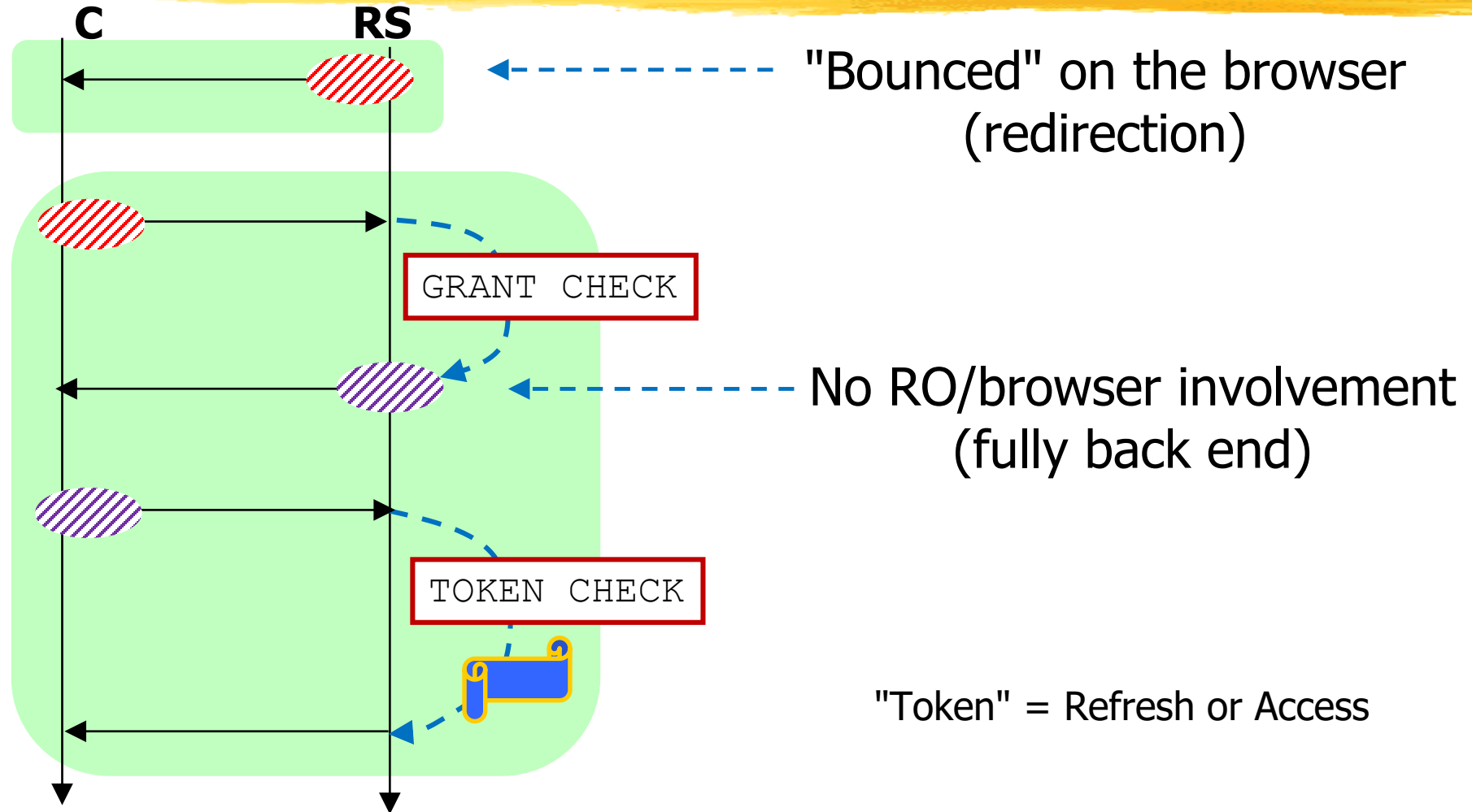
More Terminology (Not my fault)



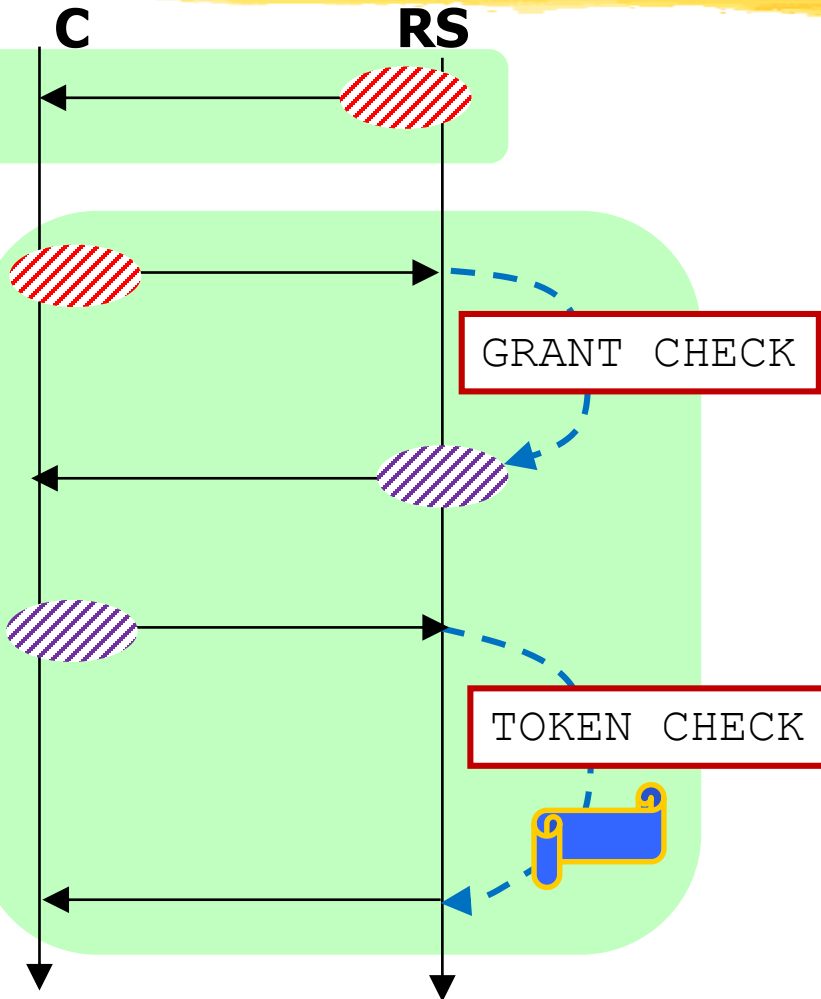
Grants and Tokens



Grants and Tokens

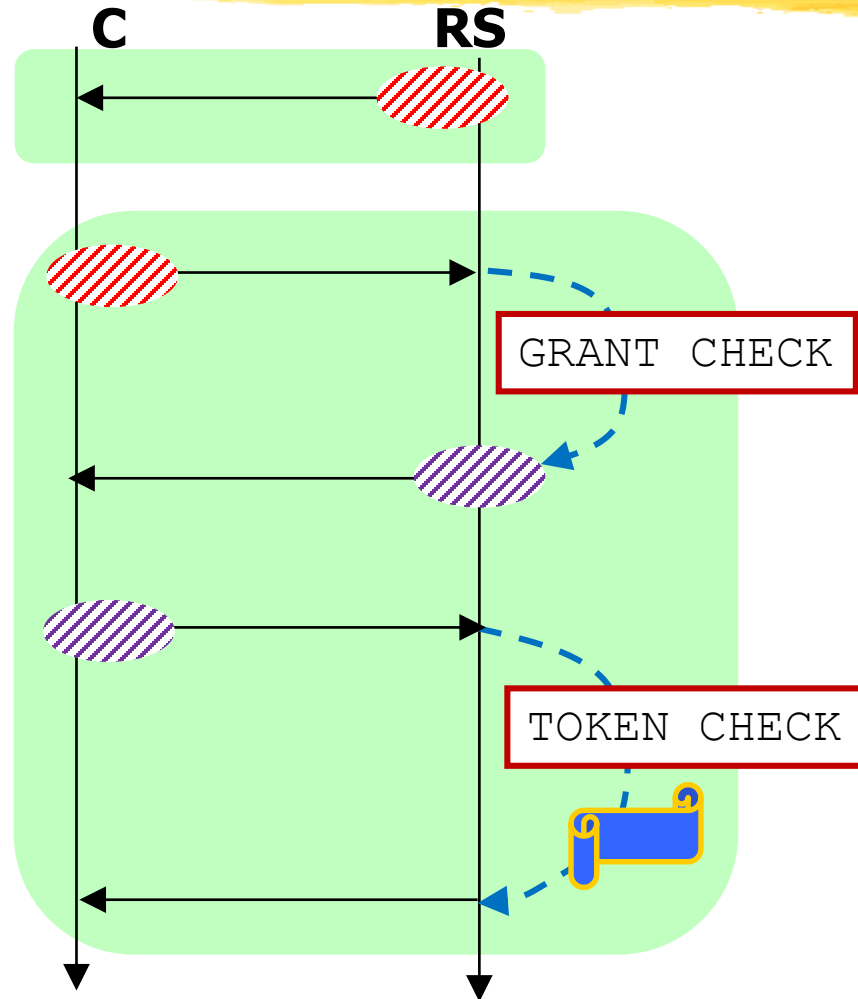


Fact



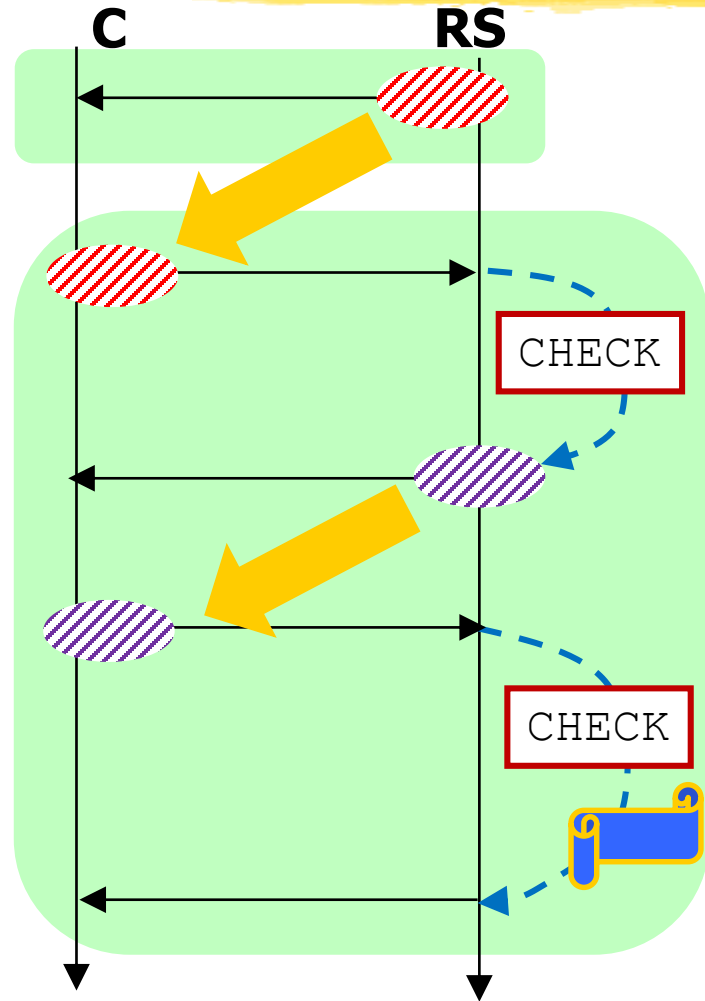
- ❑ OAuth does **not** specify the **format** of Grants/Tokens
- ❑ Every RS can choose the format that it wants
- ❑ Clients treat grants/tokens as opaque
- ❑ We will see common implementations

Requirements

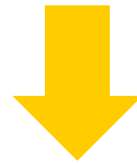


- ☐ Authentication
 - ☐ Integrity
 - ☐ Not expired / Not revoked
 - ☐ Presented by its Owner
-
- ☐ Which **resource**
 - ☐ Which **operations**

Key Remark



□ Grants and Tokens are **issued** and **verified** by the **same** entity (RS)



□ Verifying **authenticity** and **integrity** is trivial

□ Example:

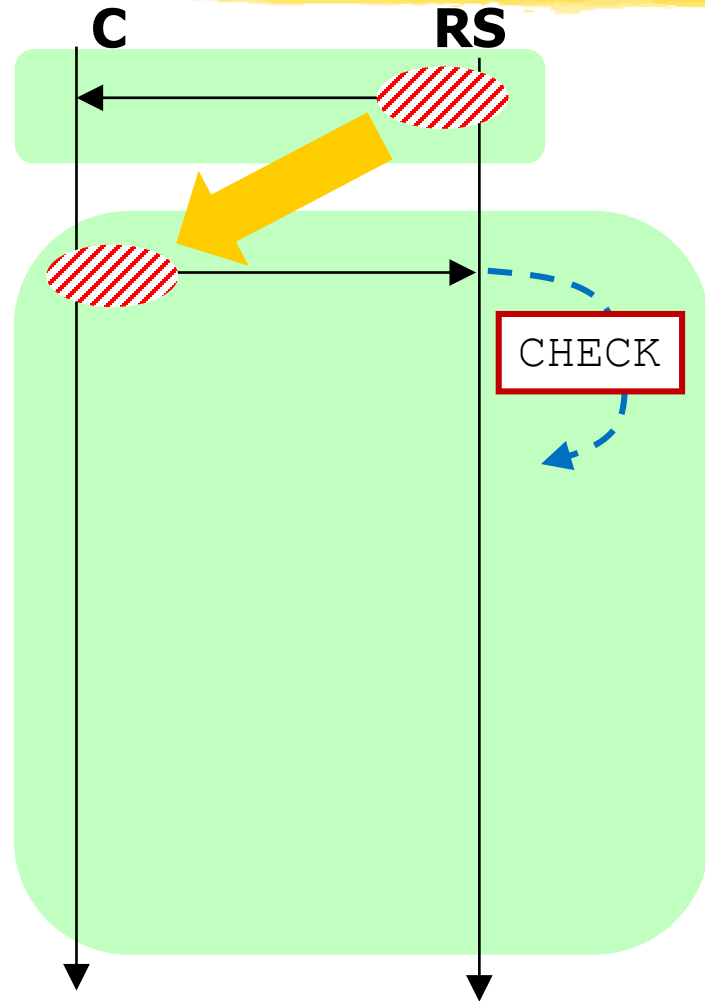
□ Issuance

→ Keep a copy

□ Presentation

→ Identical to copy?

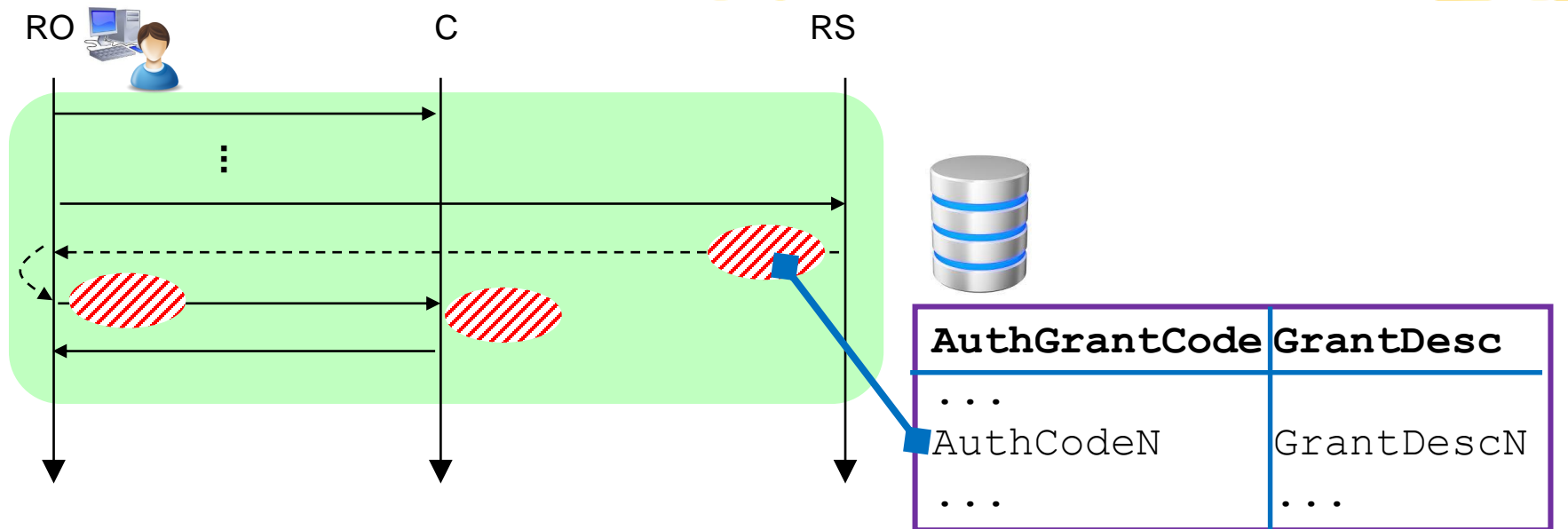
Authorization Grant: Implementation?



- ☐ Authentication
- ☐ Integrity
- ☐ Not expired / ~~Not revoked~~
- ☐ Presented by its Owner

- ☐ Which **resource**
- ☐ Which **operations**

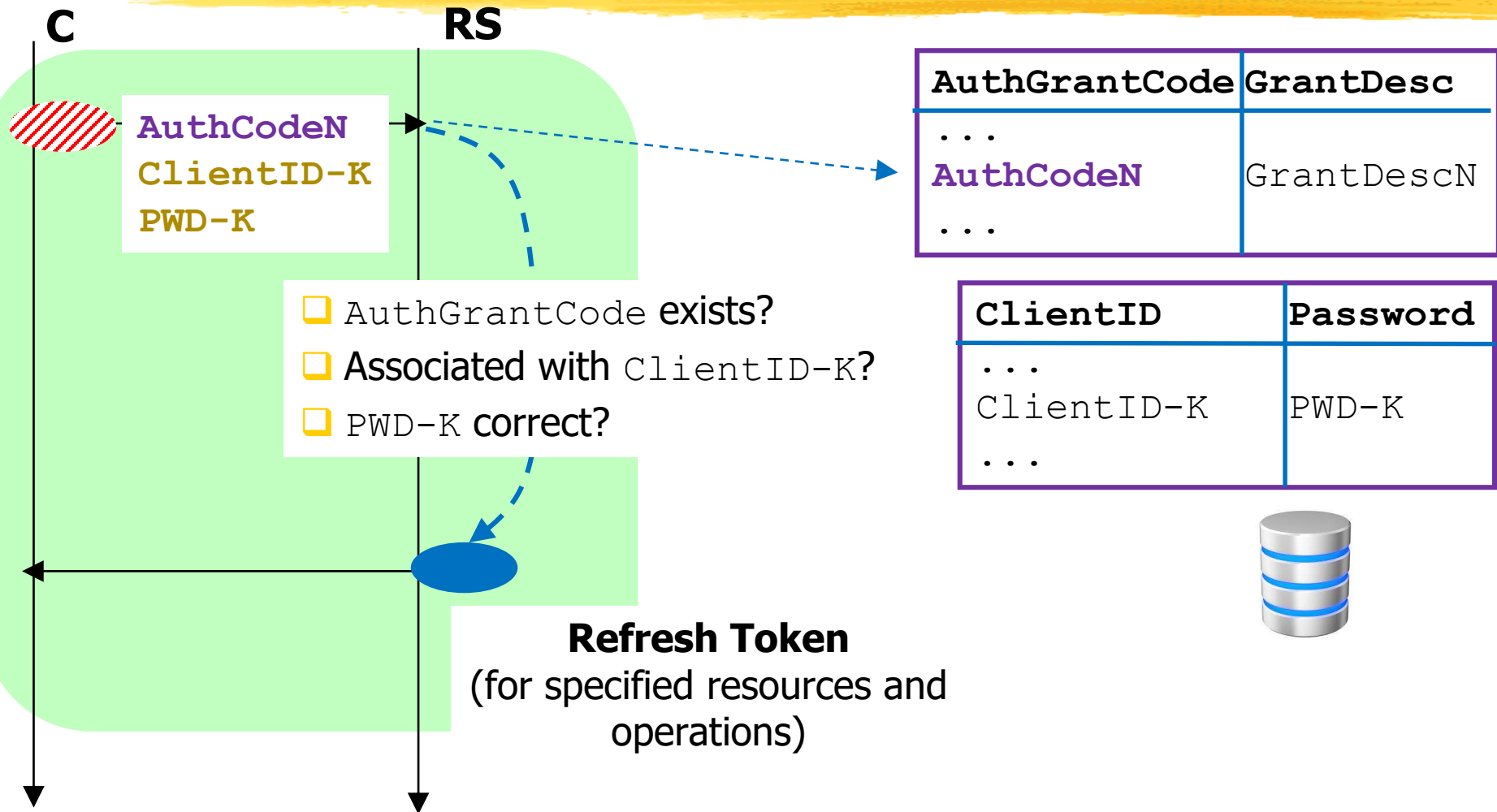
Authorization Grant: Issuance



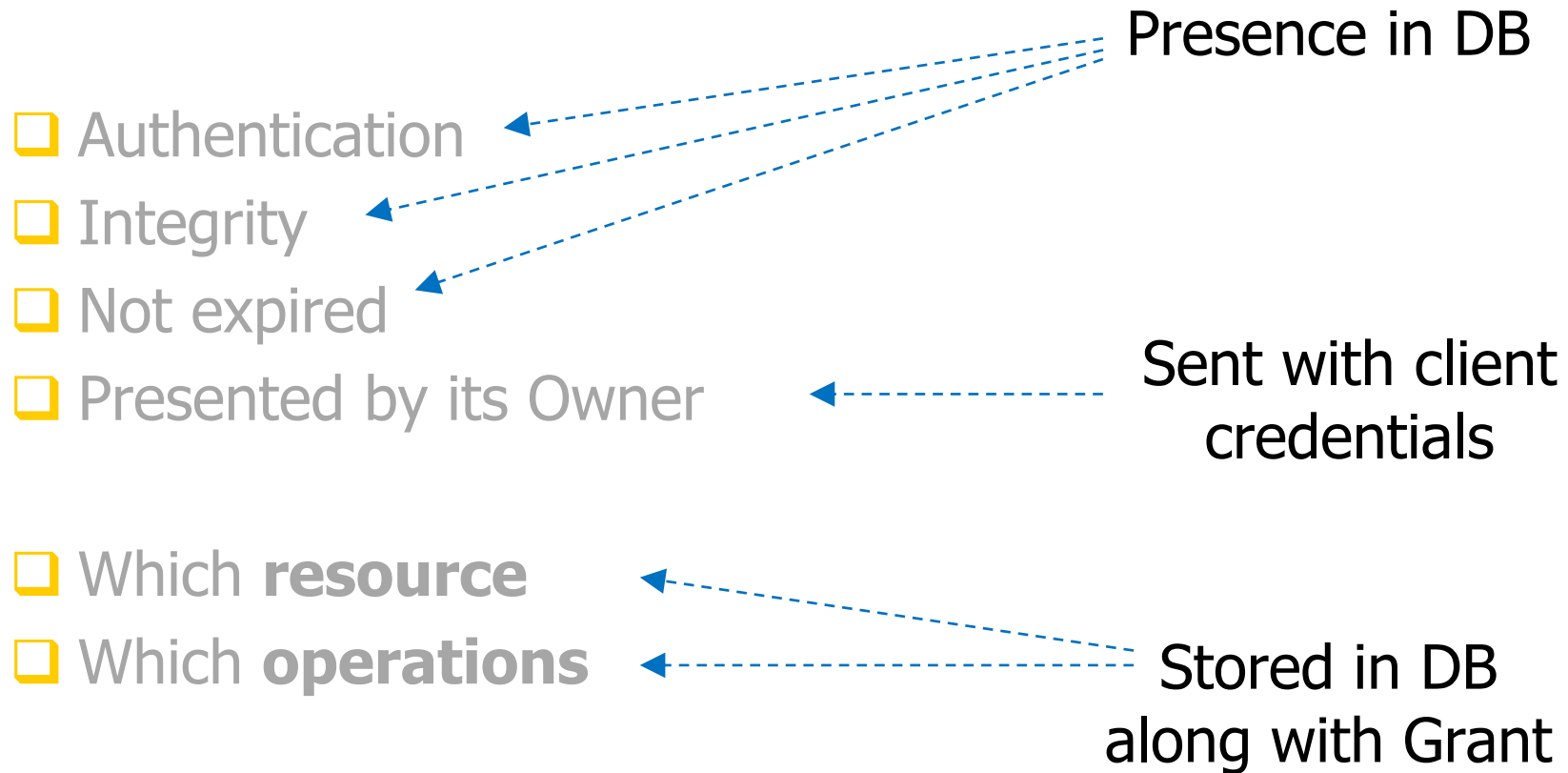
□ Authorization Code = Random-ID
(≈Cookie ID)

□ ResourceID
□ Operations
□ ClientID

Authorization Grant: Conversion



Requirements Implementation



Authorization Grant (REMINDE)



- ❑ Authorization Grant: \approx Random-ID

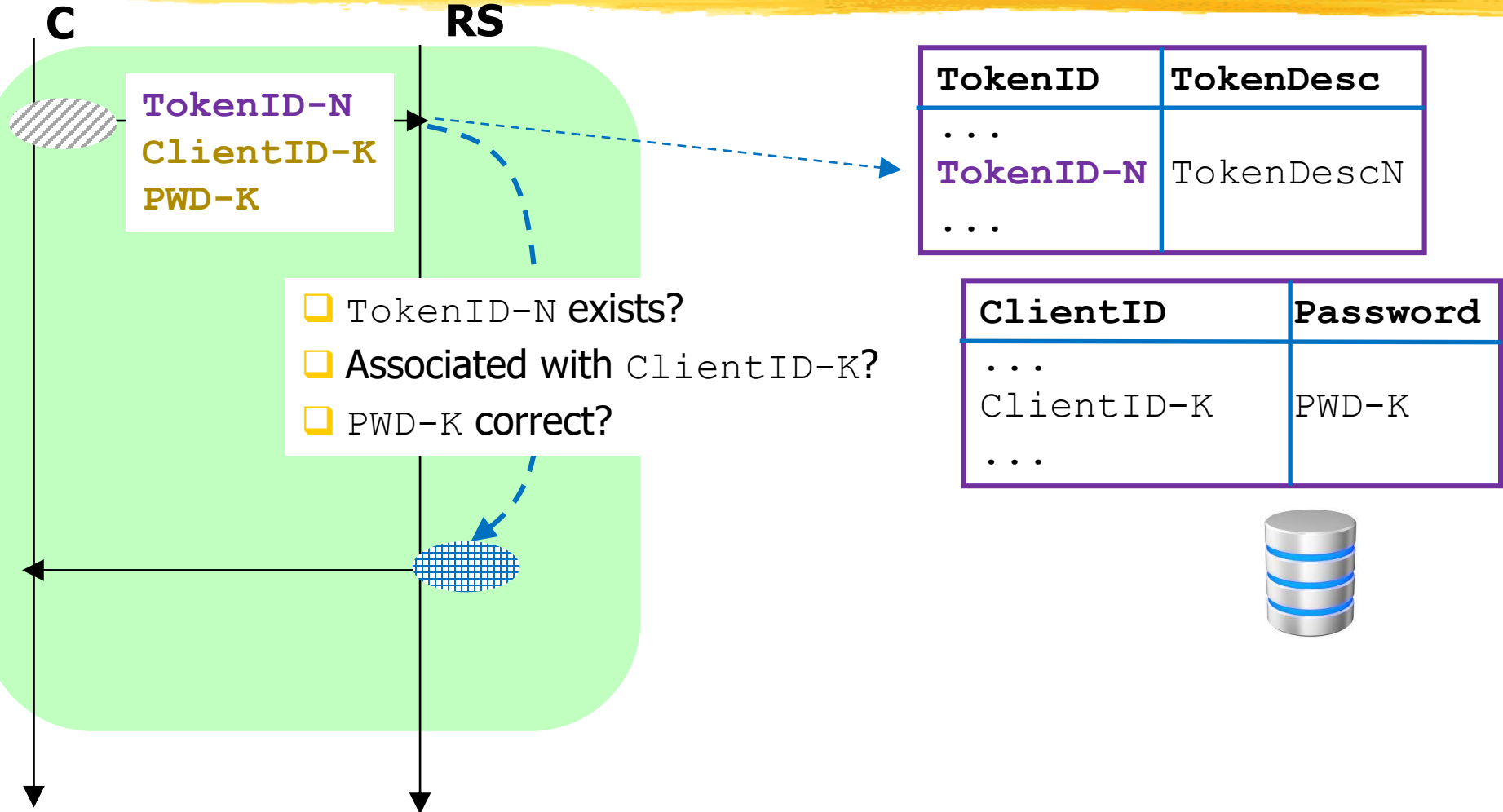
- ❑ RS database contains
 - ❑ Random-ID
 - ❑ Client Identity and Credentials of Owner
 - ❑ Resources and Access Rights
 - ❑ Time validity

Token



- ❑ **Token** \approx **Random-ID**
- ❑ RS database contains
 - ❑ **Random-ID**
 - ❑ Client Identity and Credentials of Owner
 - ❑ Resources and Access Rights
 - ❑ Time validity

Token Check



Token Revocation



- ❑ Token: \approx Random-ID

- ❑ RS database contains
 - ❑ Random-ID
 - ❑ Client Identity and Credentials of Owner
 - ❑ Resources and Access Rights
 - ❑ Time validity





- ❑ Just remove from RS database

Stateless Token (I)

 ~~Token \approx Random-ID~~

—

 ~~RS database contains~~

-  Random-ID
-  Client Identity and Credentials of Owner
-  Resources and Access Rights
-  Time validity

Token

 **Not stored in any DB at RS**
("stateless")

Hmmm...

- ❑ Token:
 - ❑ Random-ID
 - ❑ Client Identity and Credentials of Owner
 - ❑ Resources and Access Rights
 - ❑ Time validity
- ❑ **Not stored in any DB at RS**
("stateless")

It can be forged
...or modified



Stateless Token (II)



- ❑ Token: \approx
 - ❑ Random-ID
 - ❑ Client Identity and Credentials of Owner
 - ❑ Resources and Access Rights
 - ❑ Time validity
 - ❑ **Signature** by RS
- ❑ **Not stored in any DB at RS** ("stateless")
- ❑ Token Check:
 - ❑ RS verifies signature and trusts token content

Stateless Token (III)



- ❑ Token: \approx
 - ❑ Random-ID
 - ❑ Client Identity and Credentials of Owner
 - ❑ Resources and Access Rights
 - ❑ Time validity
 - ❑ Signature by RS

- ❑ **Not stored in any DB at RS ("stateless")**
- ❑ Token Check:
 - ❑ RS verifies signature and trusts token content

- ❑ **Cannot be revoked!**

Many "hybrids"



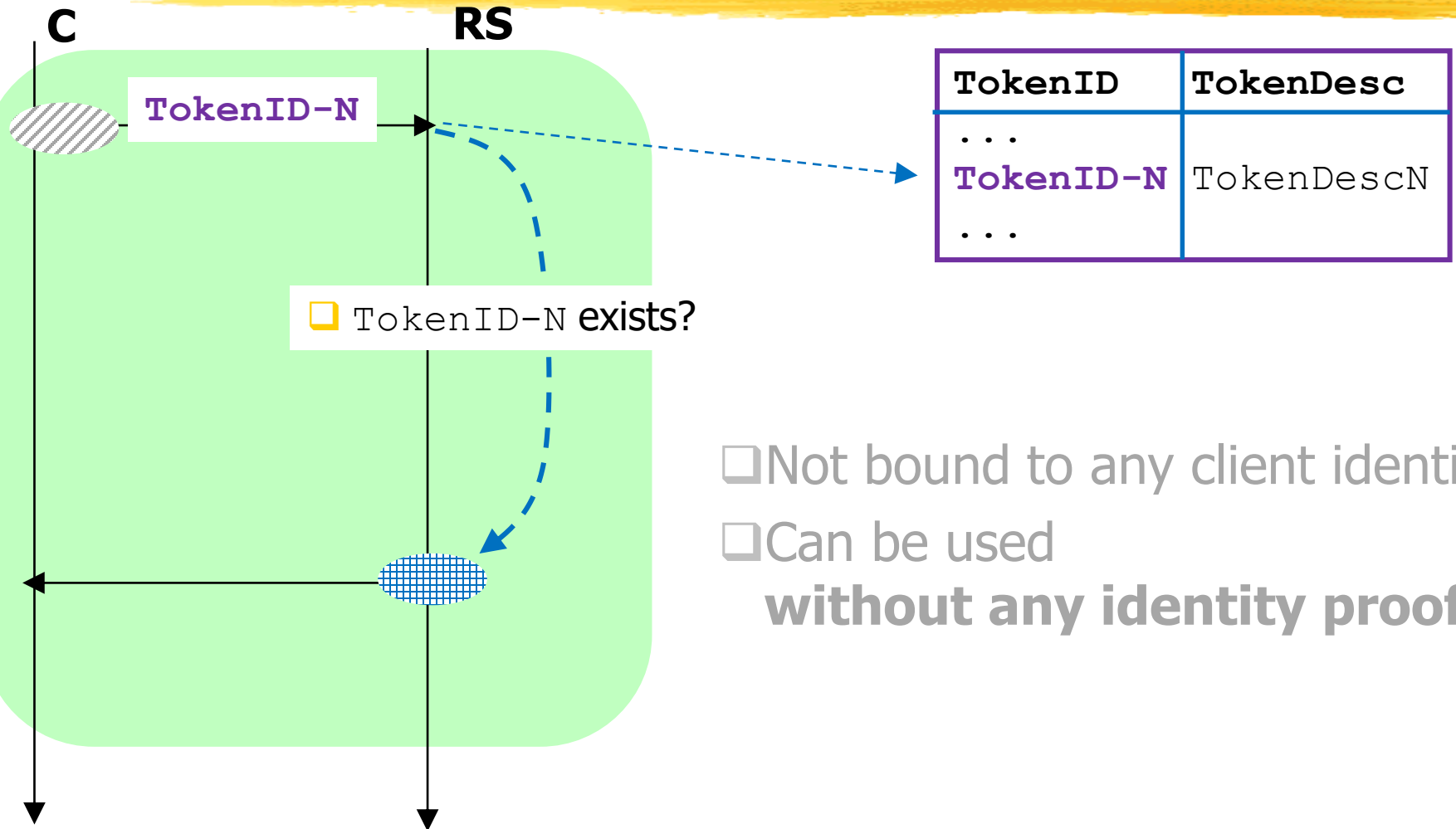
- ❑ Token: \approx
 - ❑ Random-ID
 - ❑ Client Identity and Credentials of Owner
 - ❑ Resources and Access Rights
 - ❑ Time validity
 - ❑ Signature by RS
- ❑ Random-ID stored in **Revocation DB** at RS
- ❑ Can be revoked but is no longer stateless

Bearer Token (I)



- ❑ Token: \approx Random-ID
- ❑ RS database contains
 - ❑ Random-ID
 - ~~❑ Client Identity and Credentials of Owner~~
 - ❑ Resources and Access Rights
 - ❑ Time validity
- ❑ Not bound to any client identity
- ❑ **Can be used without any identity proof**
- ❑ Security / Simplicity tradeoff

Bearer Token Check



- ☐ Not bound to any client identity
- ☐ Can be used **without any identity proof**

Bearer Token (II)



- ❑ Token: \approx
 - ❑ Random-ID
 - ~~❑ Client Identity and Credentials of Owner~~
 - ❑ Resources and Access Rights
 - ❑ Time validity
 - ❑ Signature by RS
- ❑ **Not stored in any DB at RS ("stateless")**
- ❑ Stateful or **stateless**

Token Summary (I)



- ❑ Stateful or Stateless
- ❑ Bearer or Bound
- ❑ **Independent** properties (4 combinations)
- ❑ Different design tradeoffs
- ❑ Especially in terms of Security

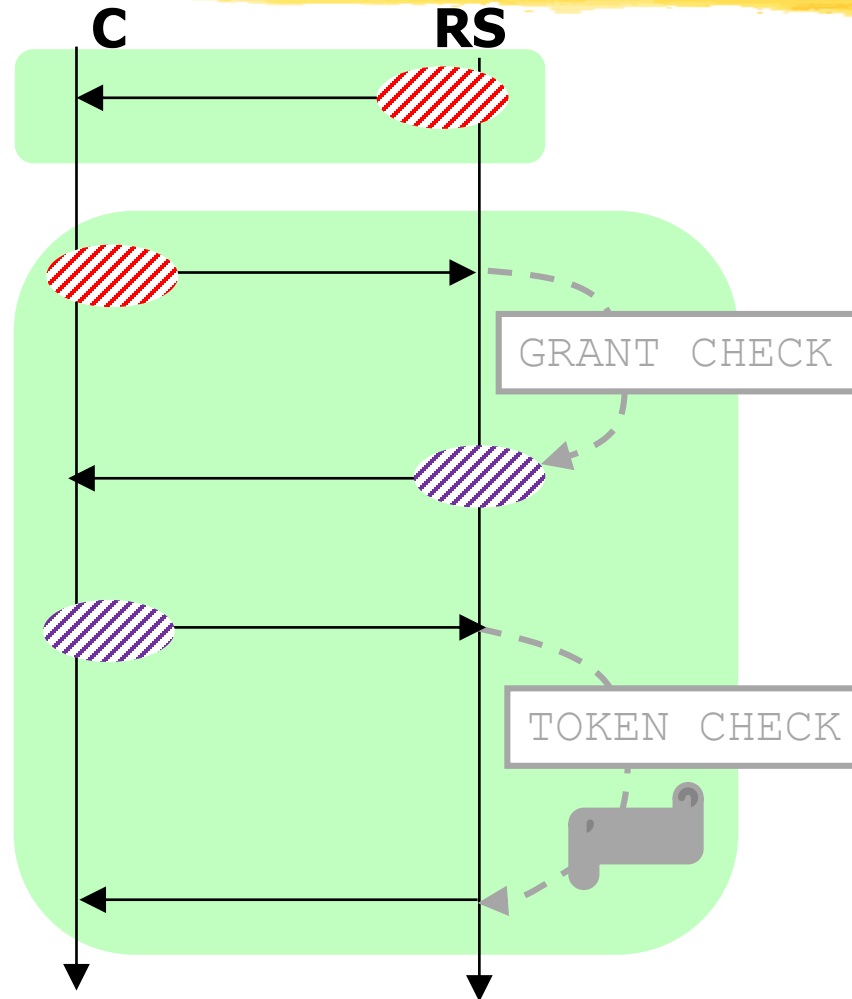
Token Summary (II)



- ❑ Bearer support:
 - ❑ Mandatory
 - ❑ Often default
 - ❑ Often used by all clients

- ❑ Can you imagine the consequence?

Representation?



- ❑ OAuth does **not** specify the **format** of Grants/Tokens
- ❑ Every RS can choose the format that it wants
- ❑ Clients treat grants/tokens as opaque
- ❑ How they are **represented** in practice?
 - ❑ Travel within HTTP traffic
 - ❑ Handled by web clients/servers

JWT: JSON Web Token

- ❑ JSON (JavaScript Object Notation) :
 - ❑ Open standard format
 - ❑ Human-readable **text** to transmit **attribute–value pairs**
 - ❑ Many libraries
- ❑ JWT have RS-specific structure and meaning (as any other token)

```
"access_token":  
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJEUEExSST  
VUTEVNMjFTQzNER0xHUjBJOFpYIiwiaXNzIjoiaHR0cHM6Ly9hcGkuc3  
RvcmlwYXR0LmNvbS92MS9hcHBsaWNhdGlvbnMvNWpvQVVKdFZONHNkT3  
rj5LATalHYa3droYkY",  
"res_id": "e3457285-b604-4990-b902-960bcadb0693",  
"scope": "can-read can-write",  
"when": "31 August 2024, 09:07:00 GMT",  
"expires_in": "3600"
```

OAuth: HTTP Usage



I'll say it again



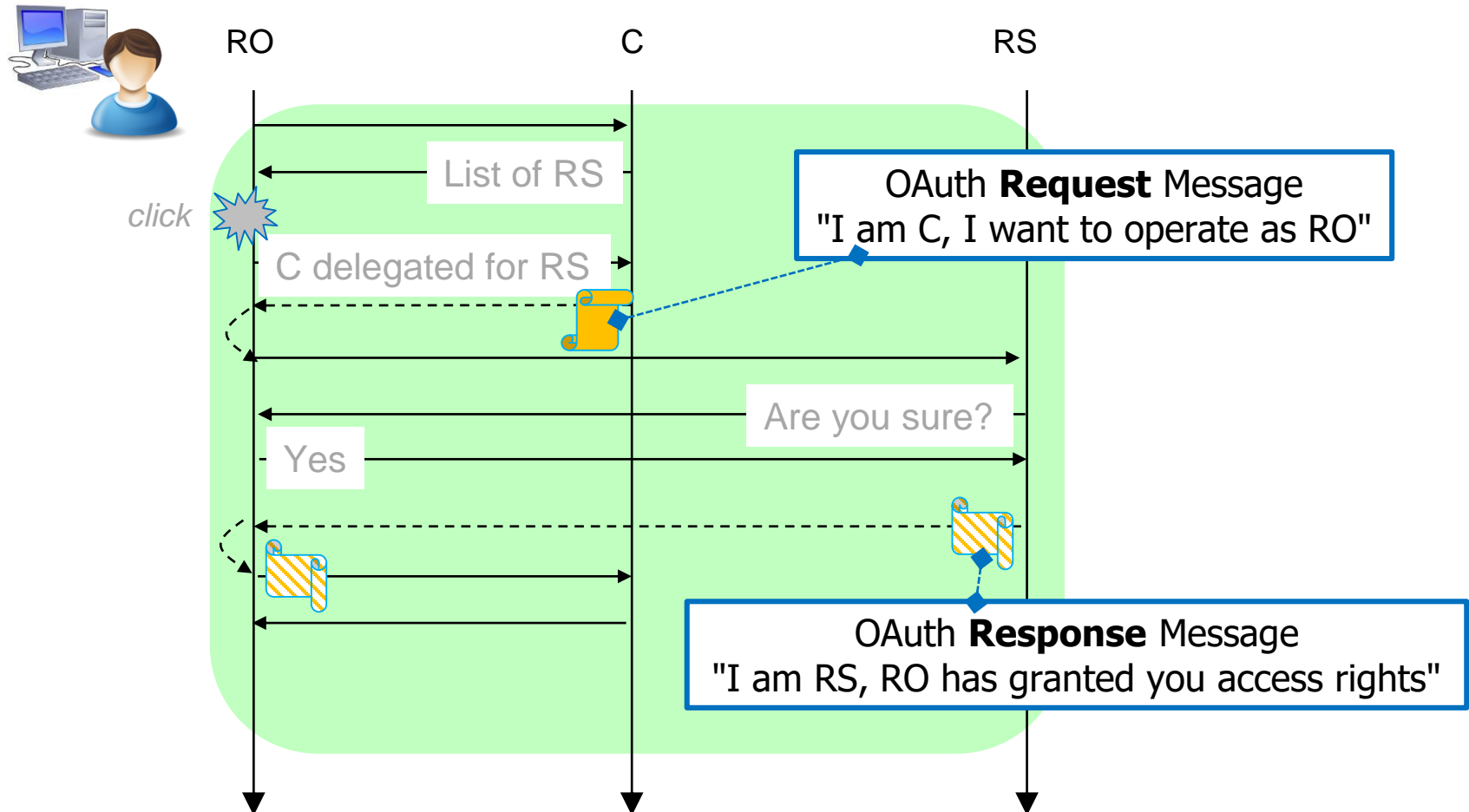
- ❑ OAuth:
 - ❑ Composed of a **lot** of **variants**
(really a lot)
 - ❑ Includes a **lot** of **details**
(really a lot)
- ❑ I admit I do not know/remember many of them
- ❑ We will focus on a few of the most important scenario

Premise



- ❑ Everything on HTTPS
- ❑ Browser has **multiple** sessions: one with C, one with RS
- ❑ We will not write cookies for simplicity
(but they are there)
- ❑ RO must be **authenticated** on both C and RS
- ❑ If session is not authenticated then login page + credential sending
- ❑ We will not write them for simplicity

Delegation phase (REMININD)



OAuth messages



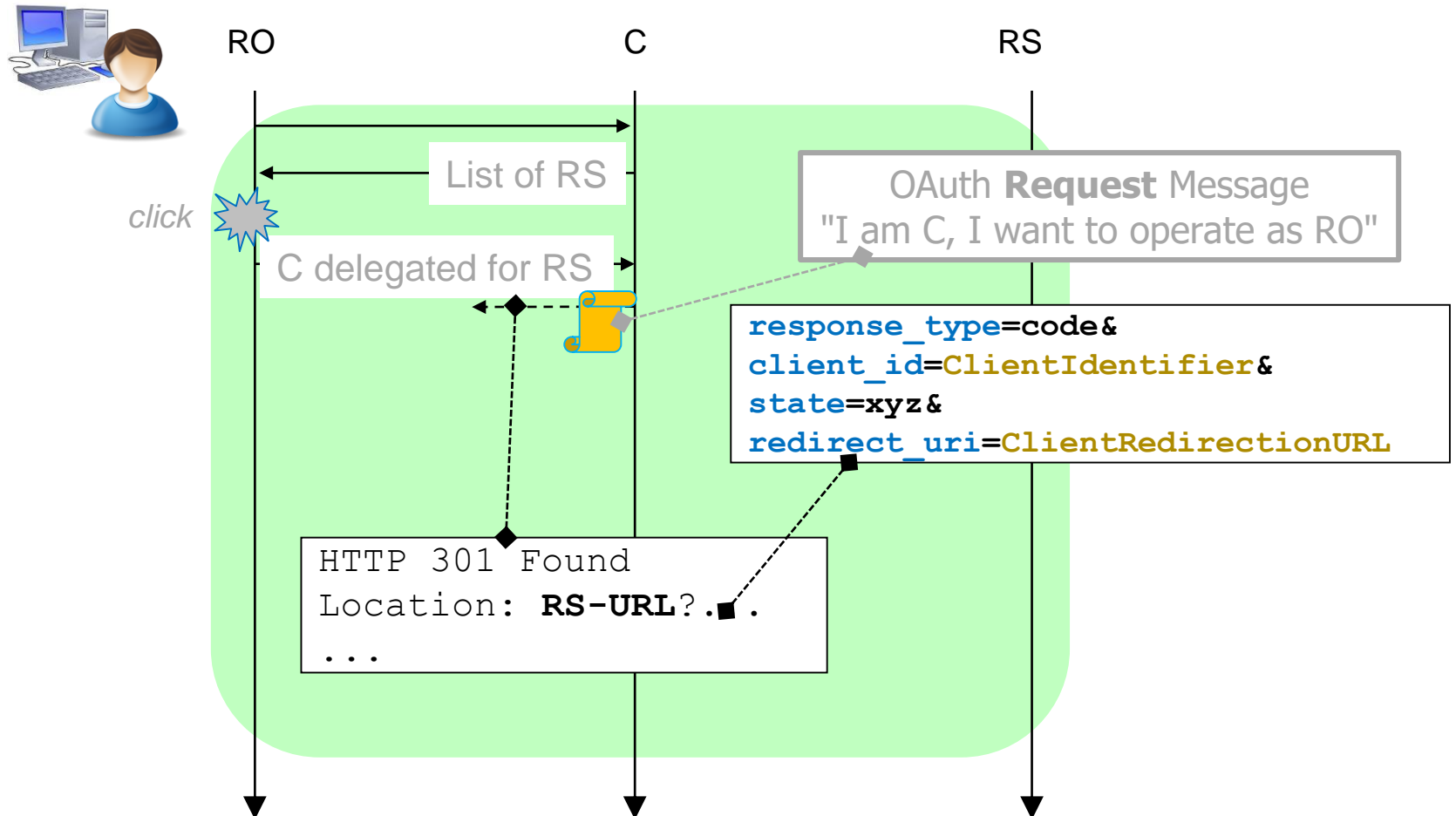
- ❑ OAuth **messages** are **query strings**

`name1=value1&name2=value2&...`

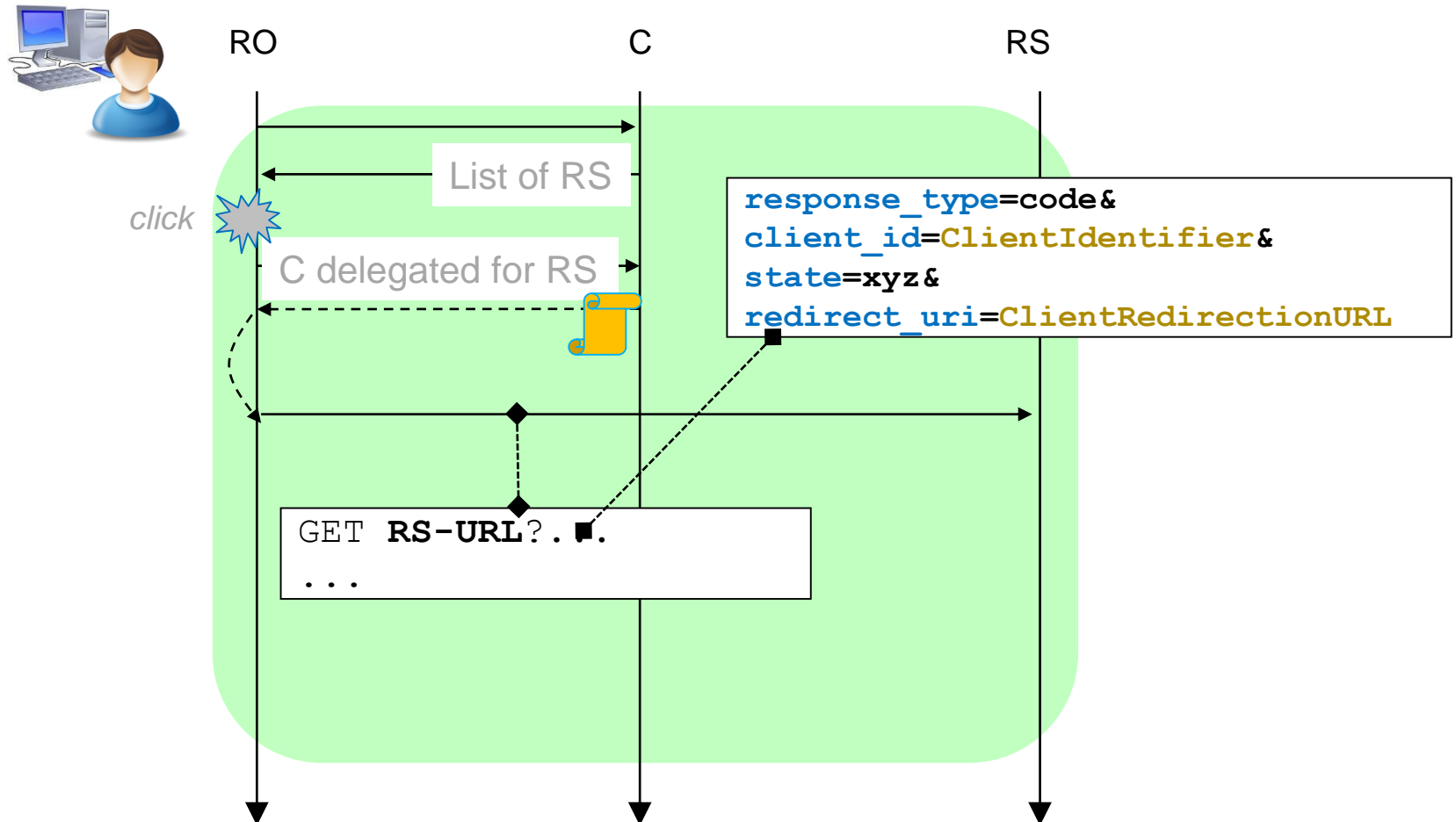
- ❑ Protocol specifies:

- ❑ Names/values that are **required** or **optional**
- ❑ How to **embed messages** in HTTP Request/Responses
- ❑ **Checks** on received messages

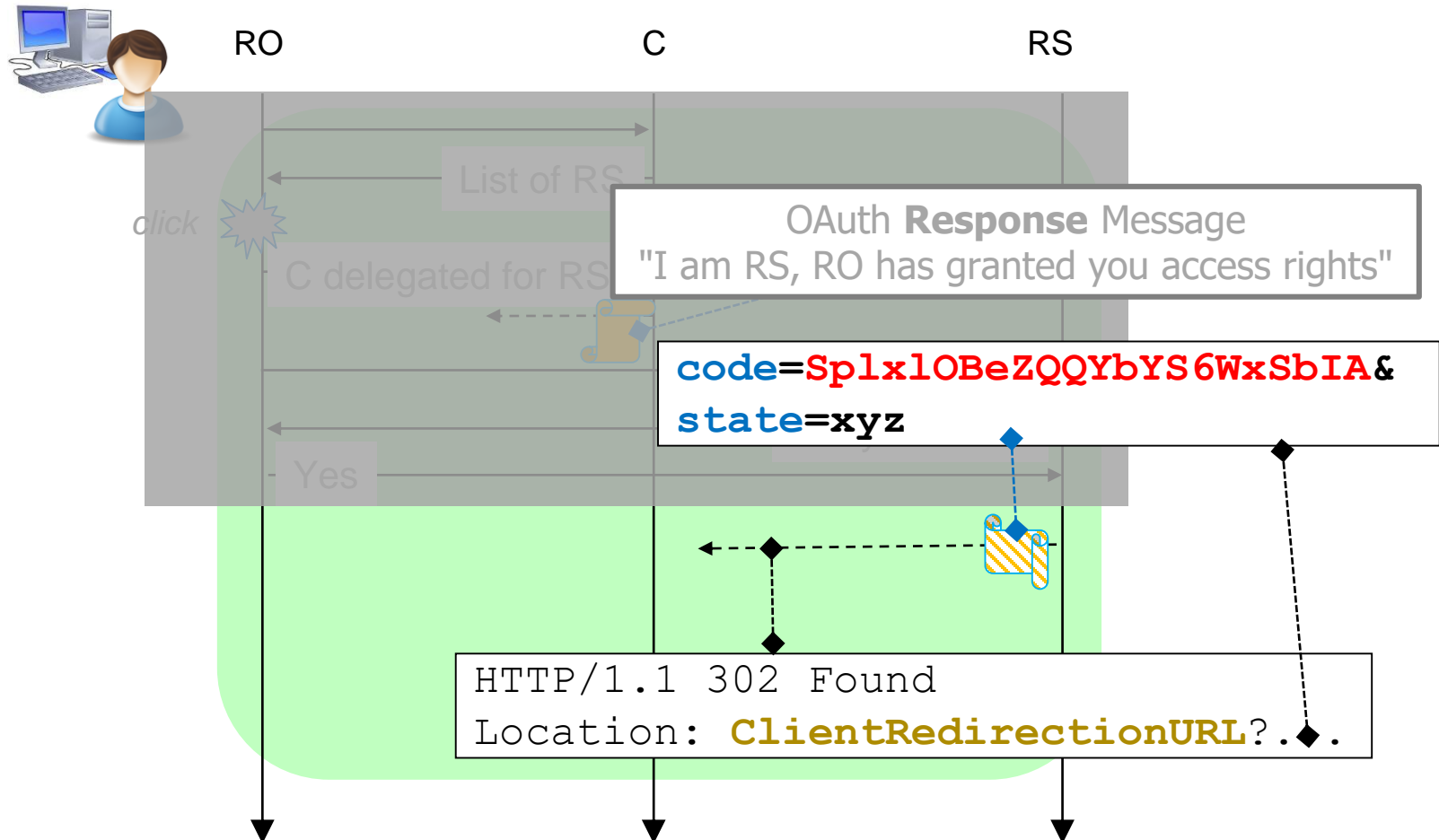
OAuth Request (I)



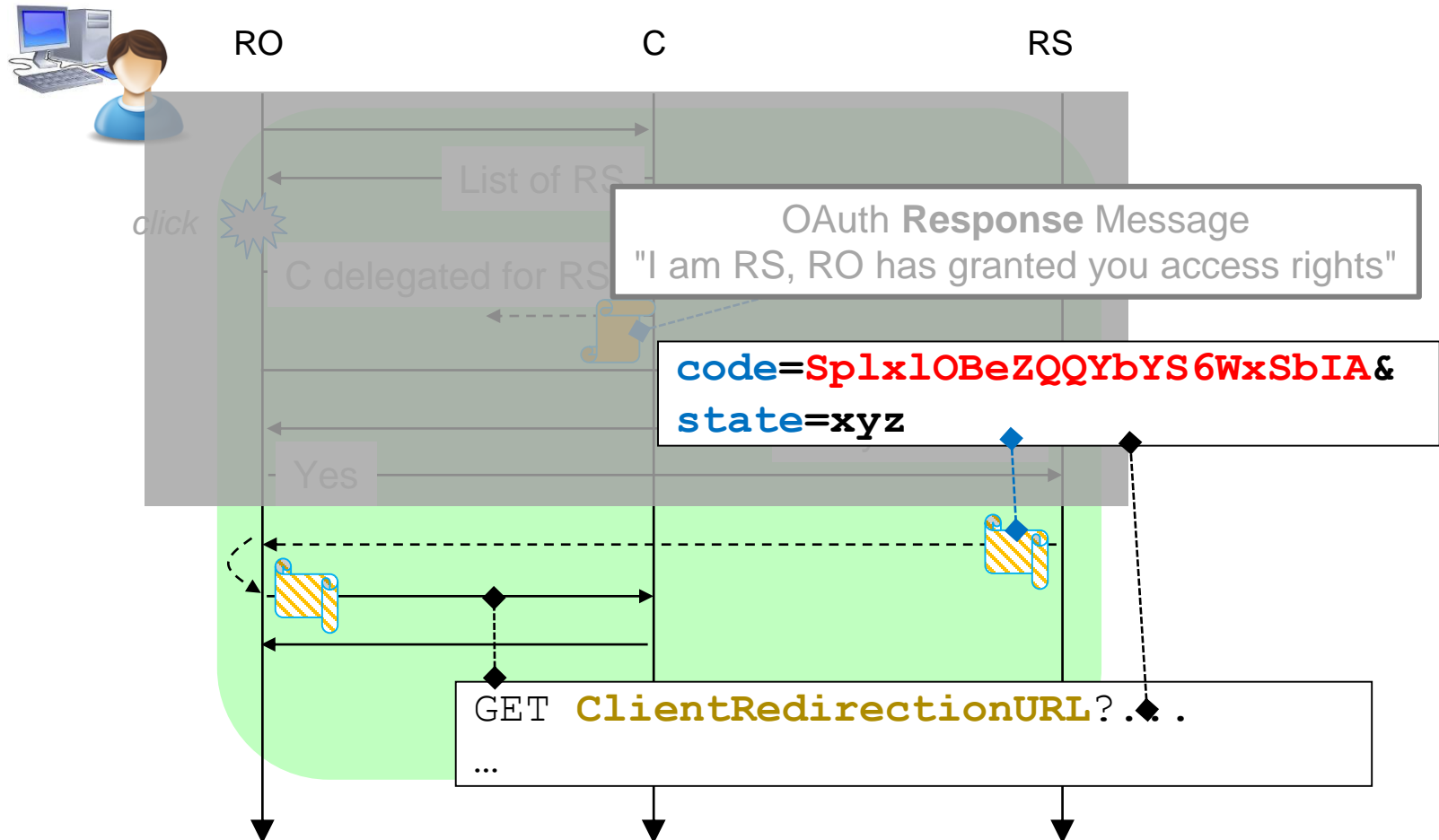
OAuth Request (II)



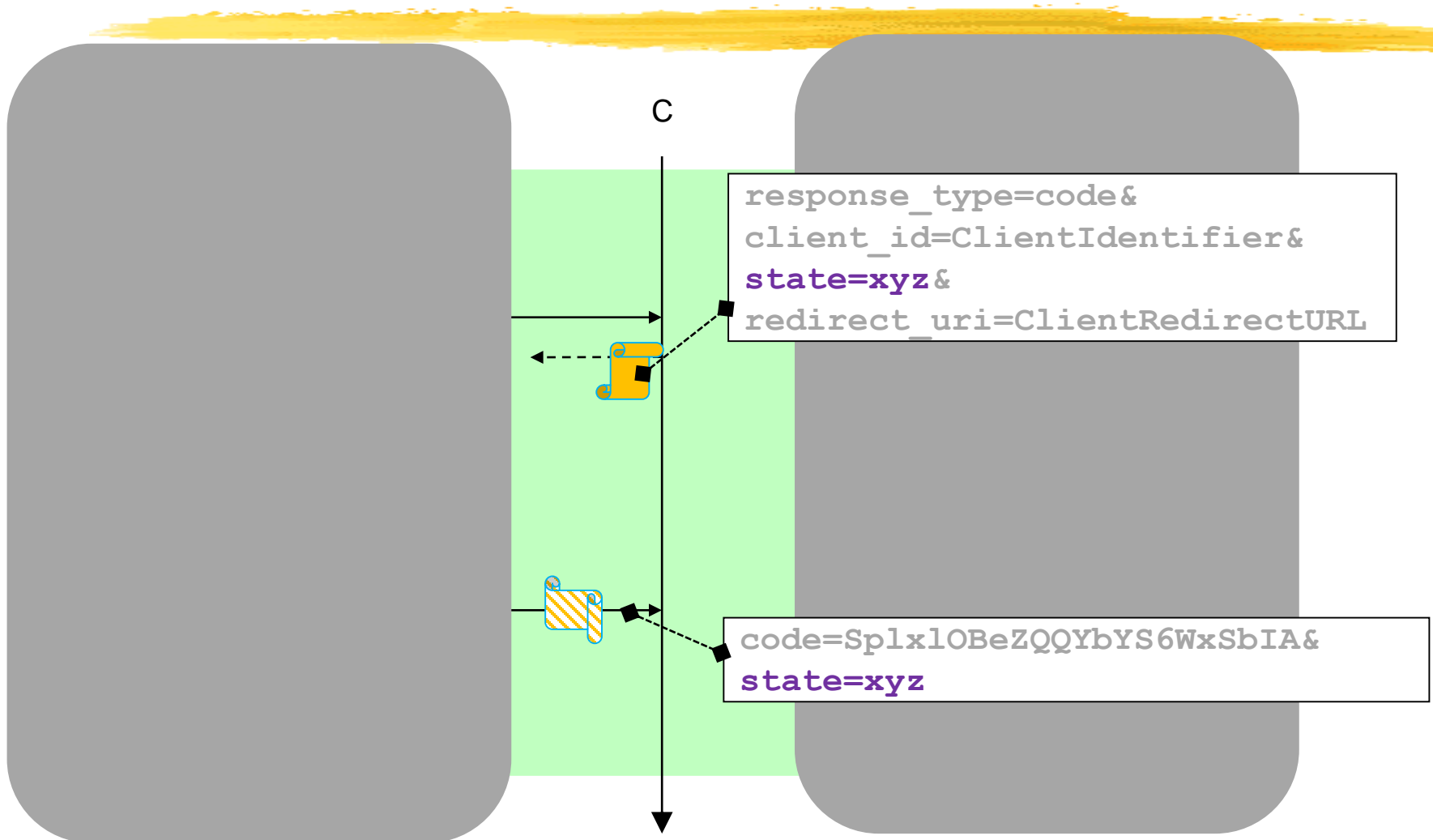
OAuth Response (I)



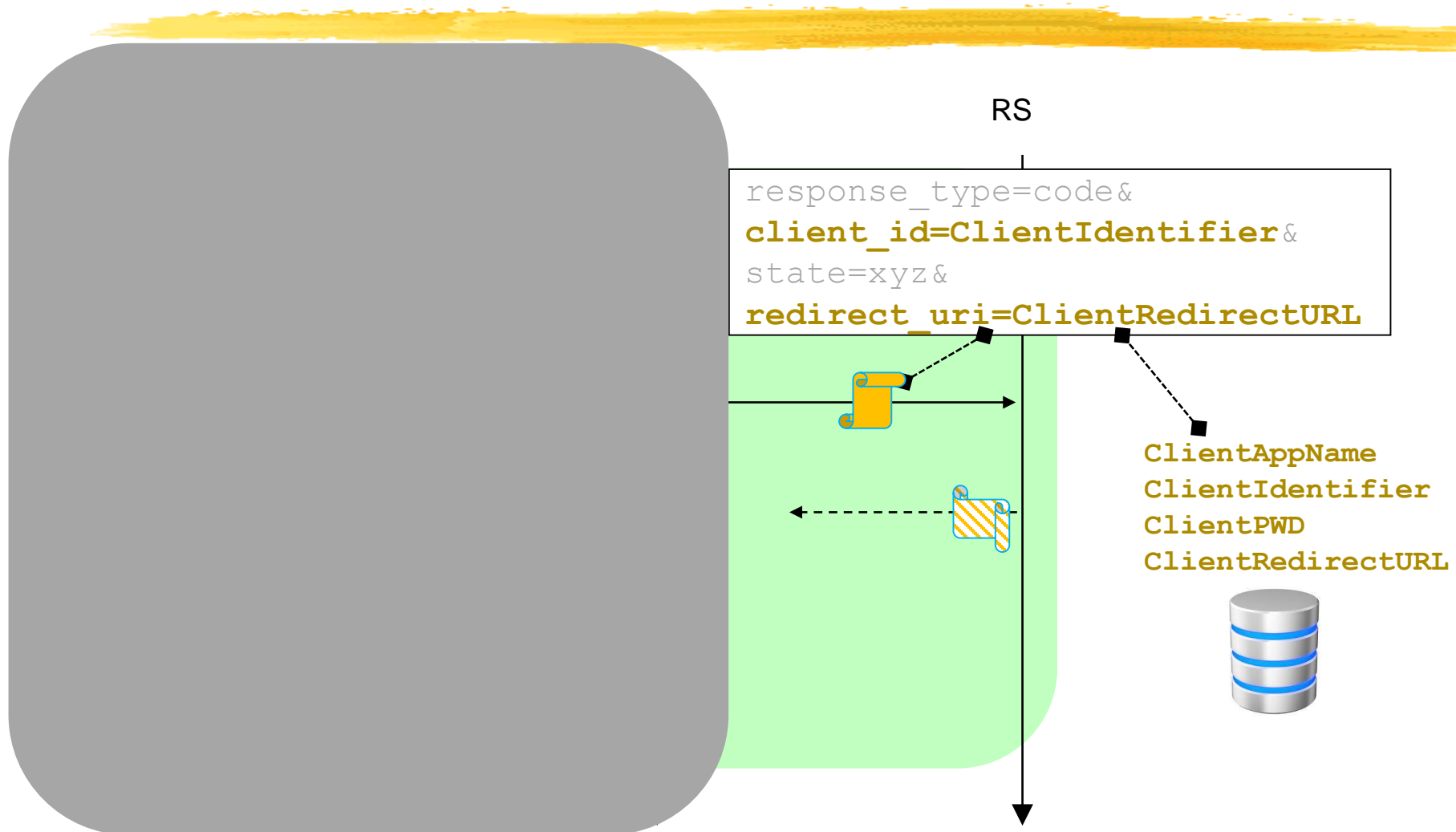
OAuth Response (II)



One of the many checks: Client



One of the many checks: Resource Server



Vulnerabilities



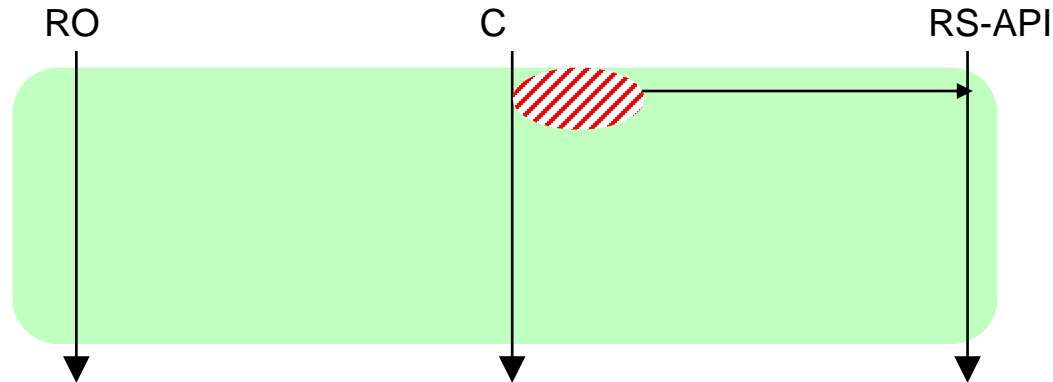
- ❑ **Vulnerabilities** in OAuth **implementations**
- ❑ Impact oversimplified:
 - ❑ GoodClient has Token(GoodClient, U@RS)
 - ❑ Vuln@GoodClient \Rightarrow
EvilClient might obtain Token(GoodClient, U@RS)
 - ❑ Several preconditions but **without U consent**
- ❑ Vuln@RS: similar impact but much rarer

Important



- ❑ OAuth specifies checks upon receiving messages
- ❑ **Vulnerabilities** in OAuth **implementations**
- ❑ **Necessary** condition for "all" such vulns:
 - ❑ Missing check
 - ❑ Wrong check
 - ❑ Bypassed check
- ❑ Never trust your input: remember?

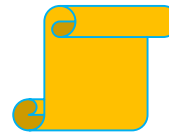
Access: Grant → Refresh Token



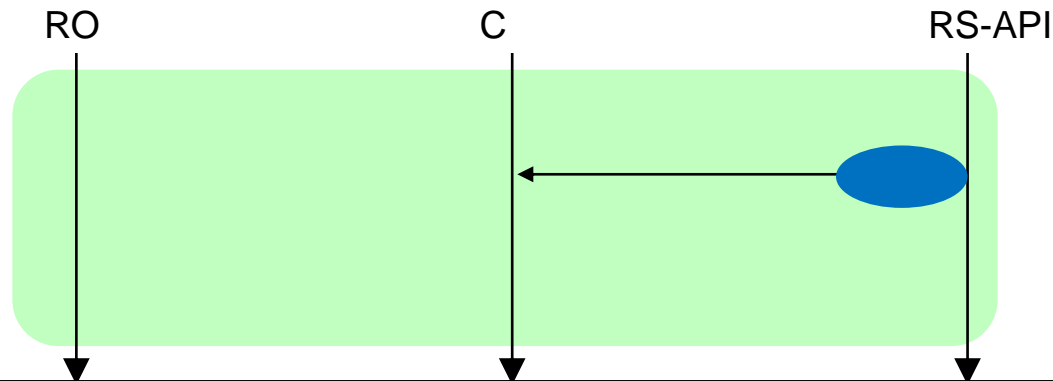
```
POST ...
Host: ...
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=authorization_code&
code=Sp1x10BeZQQYbYS6WxSbIA&
redirect_uri=ClientRedirectionURL
```

ClientID
ClientPassword



Refresh Token: Bearer Stateful



```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
```

```
{ "token_type" = "Bearer",
  "refresh_token": "2YotnFZFEjr1zCsicMWpAA",
  "expires_in": 3600
}
```



*Resources, Access Rights
in RS-DB*

Refresh Token: Bearer Stateless

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "refresh_token":
  "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJEUEExS
  STVUTEVNMjFTQzNER0xHUjBJOFpYIiwiaXNzIjoiaHR0cHM6Ly9hcG
  kuc3RvcmlwYXR0LmNvbS92MS9hcHBsaWNhdGlbnMvNWpvQVVKdFZO
  NHNkT3rj5LATalHYa3droYkY",
  "res_id": "e3457285-b604-4990-b902-960bcadb0693",
  "scope": "can-read can-write",
  "when": "31 August 2023, 09:07:00 GMT",
  "expires_in": "3600"
}
```

*Resources, Access Rights
in RS-DB*

Refresh → Access

```
POST ...
Host: ...
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&
      refresh_token=2YotnFZFEjr1zCsicMWpAA&
      code=Sp1x10BeZQQYbYS6WxSbIA&
      redirect_uri=ClientRedirectionURL
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{ "token_type" = "Bearer",
  "access_token": "2Yot6789489njh0kzCsicp31",
  "expires_in": 3600
}
```

Access API

❑ Highly RS-dependent

❑ Client is not a Browser → Many ways for sending access tokens

❑ Bearer token:

GET/POST ...

Authorization: Bearer **2YotnFZFEjr1zC...sicMWpAA**

...

*Stateless
or Stateful (JWT + Base64)*

❑ Bearer / Bound Token:

GET/POST ...

Content-Type: application/json

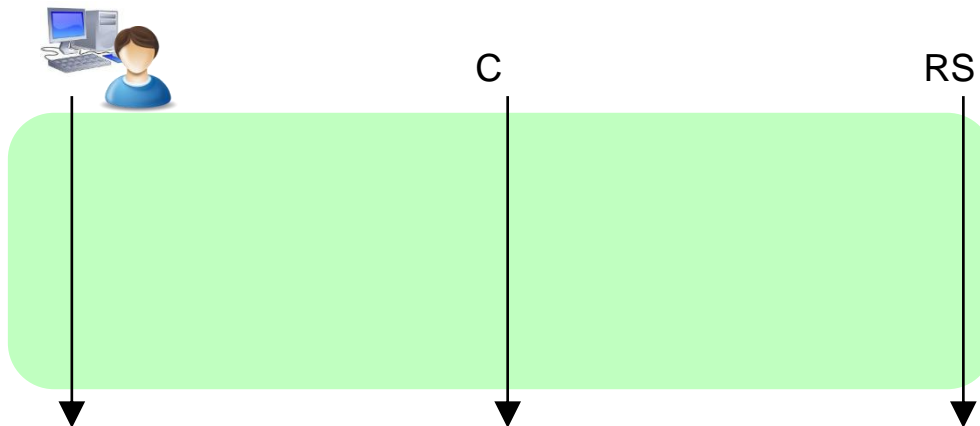
...

{ ... }

*Stateless
or Stateful (JWT)*

Practical Considerations (I)

- ❑ OAuth is more of a **framework** than a **protocol**
 - ❑ OAuth implementations by different orgs are "**not very likely**" to be interoperable
- ↓
- ❑ C must be written with **RS-provided** OAuth library



Practical Considerations (II)



Using OAuth 2.0 to Access Google APIs



OAuth 2.0

Live Connect implements the [OAuth 2.0](#) protocol to authenticate users. This topic describes both the authorization flows that Live Connect uses and the supported extension parameters.



Sharing


Enable people to post to Facebook from your app.



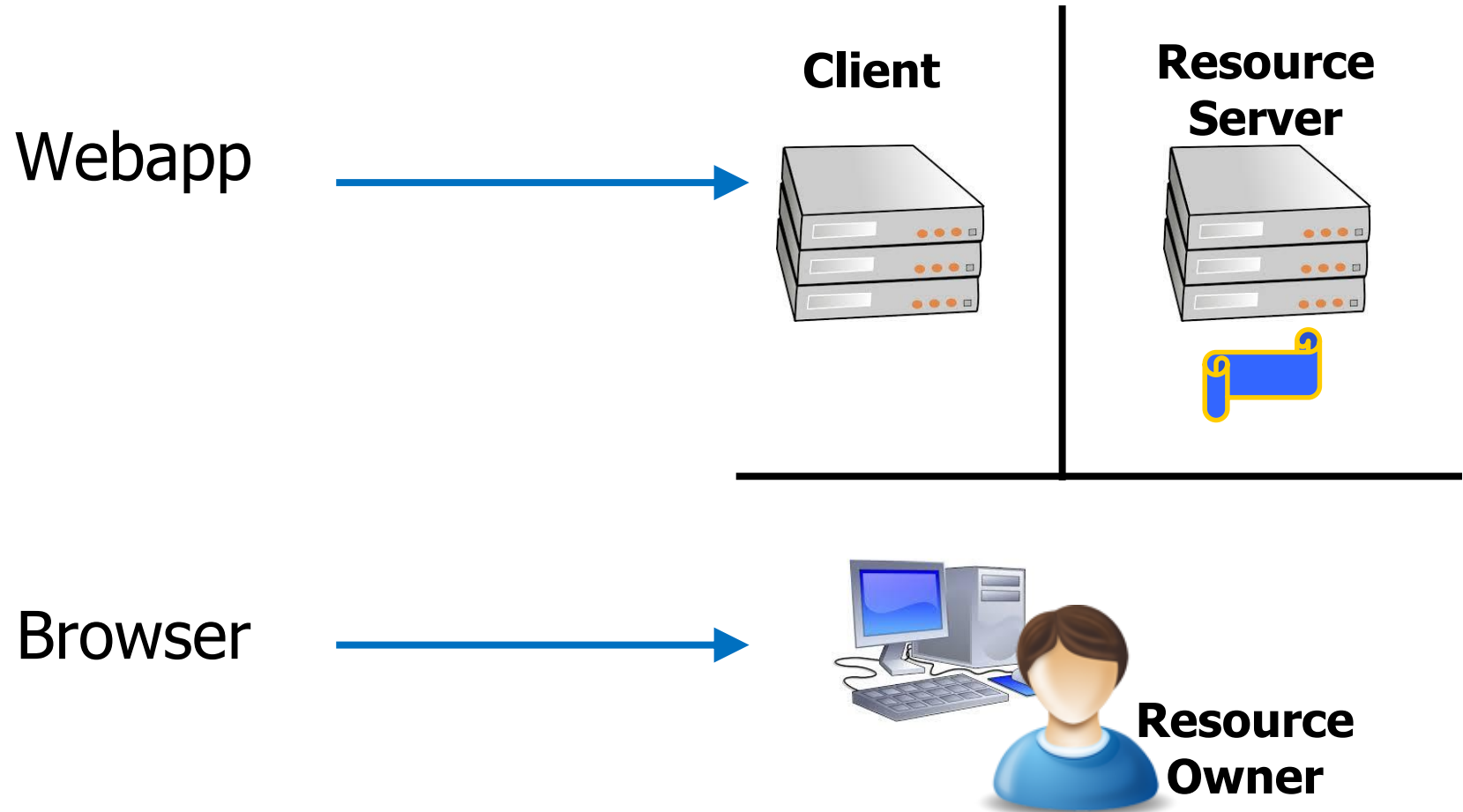
The PayPal API uses HTTP methods and a RESTful endpoint structure. The API authorization framework is [OAuth 2.0](#). You format requests in JSON and the APIs return JSON-formatted responses.

OAuth

as a general framework



REMIND



What OAuth is used for (I)



- Client Program:

- Webapp

- Program downloaded from a **public** store and executed on **PC**

- Dropbox

- Google Drive

- OneDrive

- Evernote

- ...

What OAuth is used for (II)

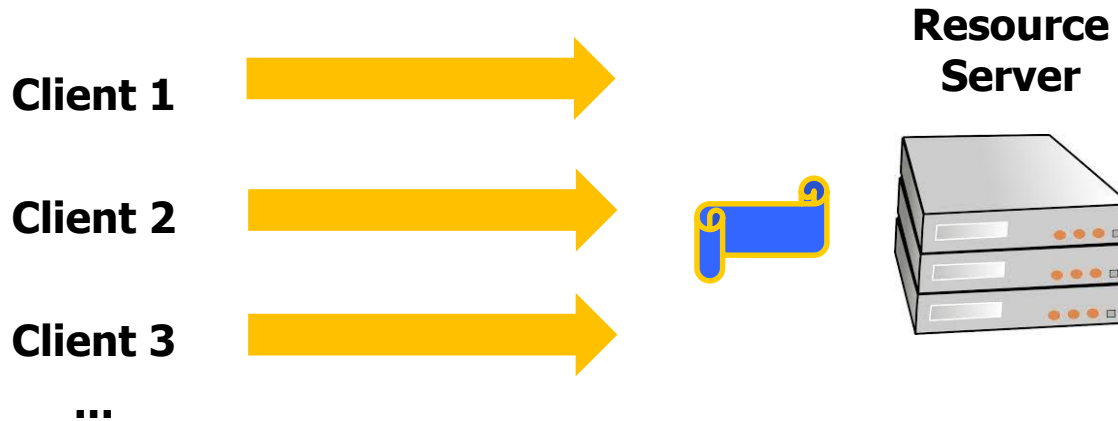


- ❑ Client Program:
 - ❑ Webapp
 - ❑ Program downloaded from a **public** store and executed on **PC**
 - ❑ App downloaded from a **public** store and executed on **Smartphone**
 - ❑ Facebook
 - ❑ Twitter
 - ❑ Unicredit
 - ❑ Postepay
 - ❑ ...

What OAuth is used for (III)

- ❑ Client Program:
 - ❑ Webapp
 - ❑ Program downloaded from a **public** store and executed on **PC**
 - ❑ App downloaded from a **public** store and executed on **Smartphone**
 - ❑ Program developed **privately** and executed on **PC**
 - ❑ My Python SlideMaster (read/write Google Spreadsheets / Google Forms)
 - ❑ Program developed **privately** and executed on **Server**
 - ❑ **Javascript App** downloaded from a web server and executed on **Browser**
 - ❑ ...

Great! (REMIND)

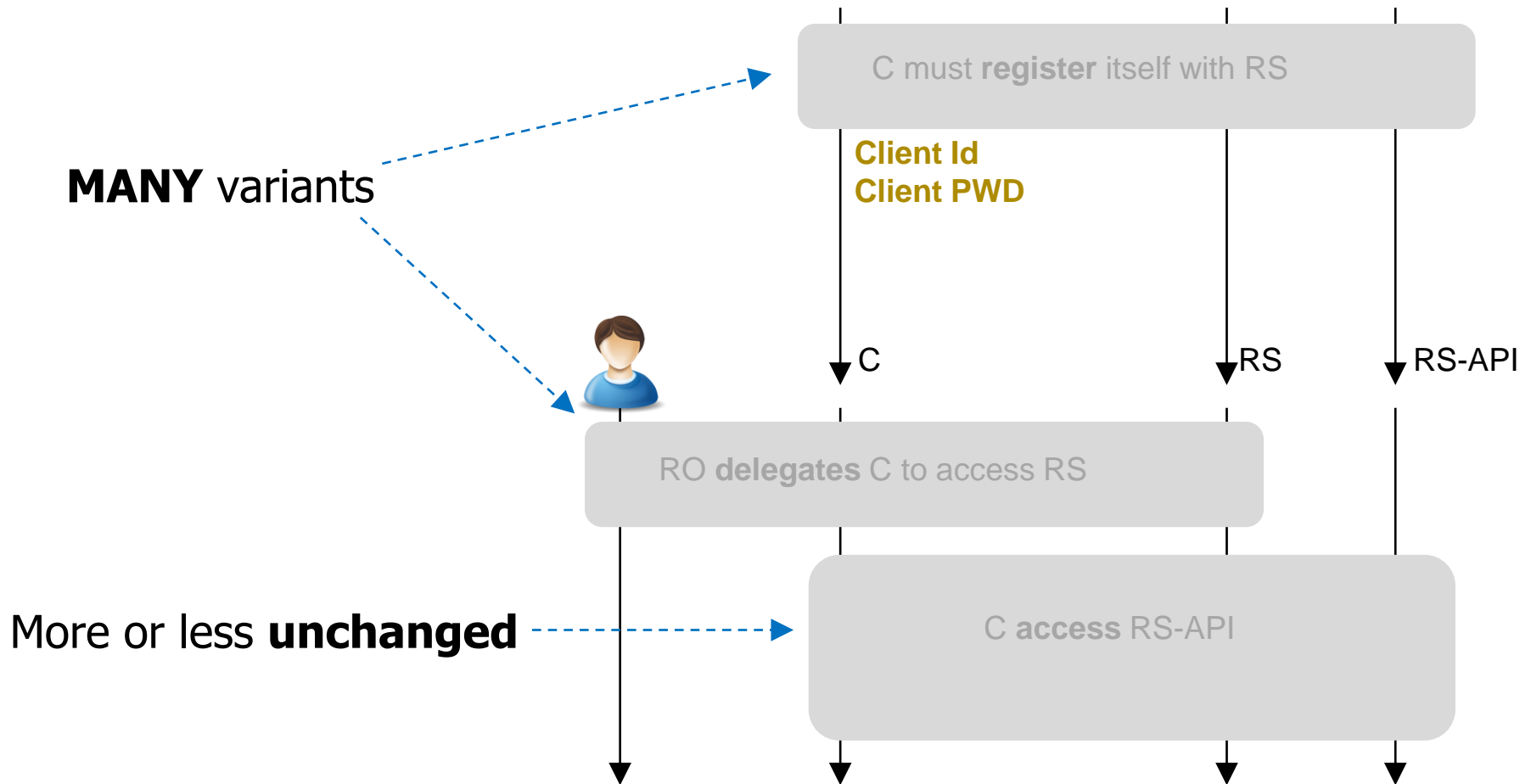


- ❑ **Any kind** of program
- ❑ ...running **anywhere**

- ❑ **Centralized** control of **all** delegations
- ❑ Can **revoke** at any time with just **one click**



Where differences are



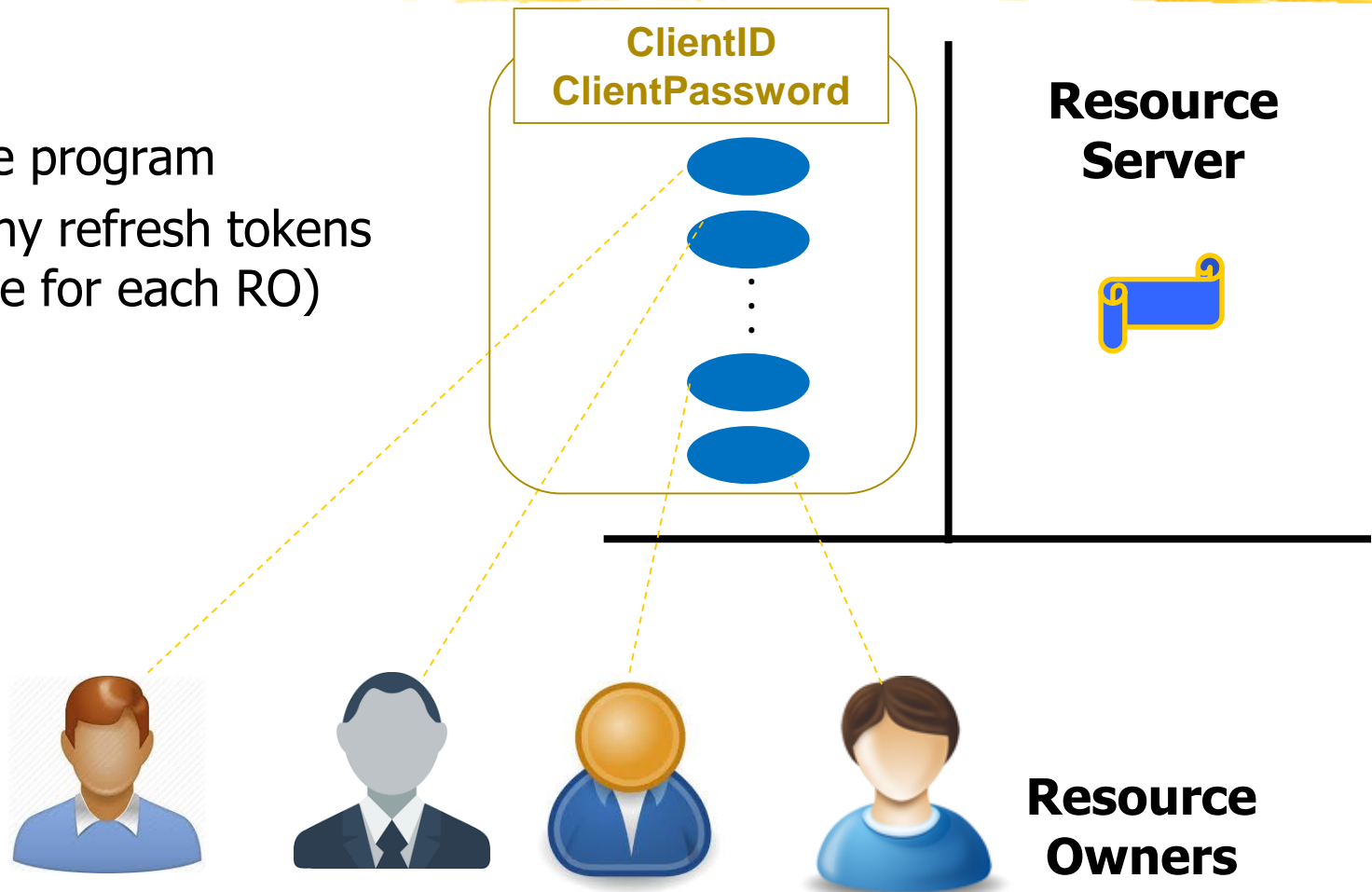
Just a few words on...



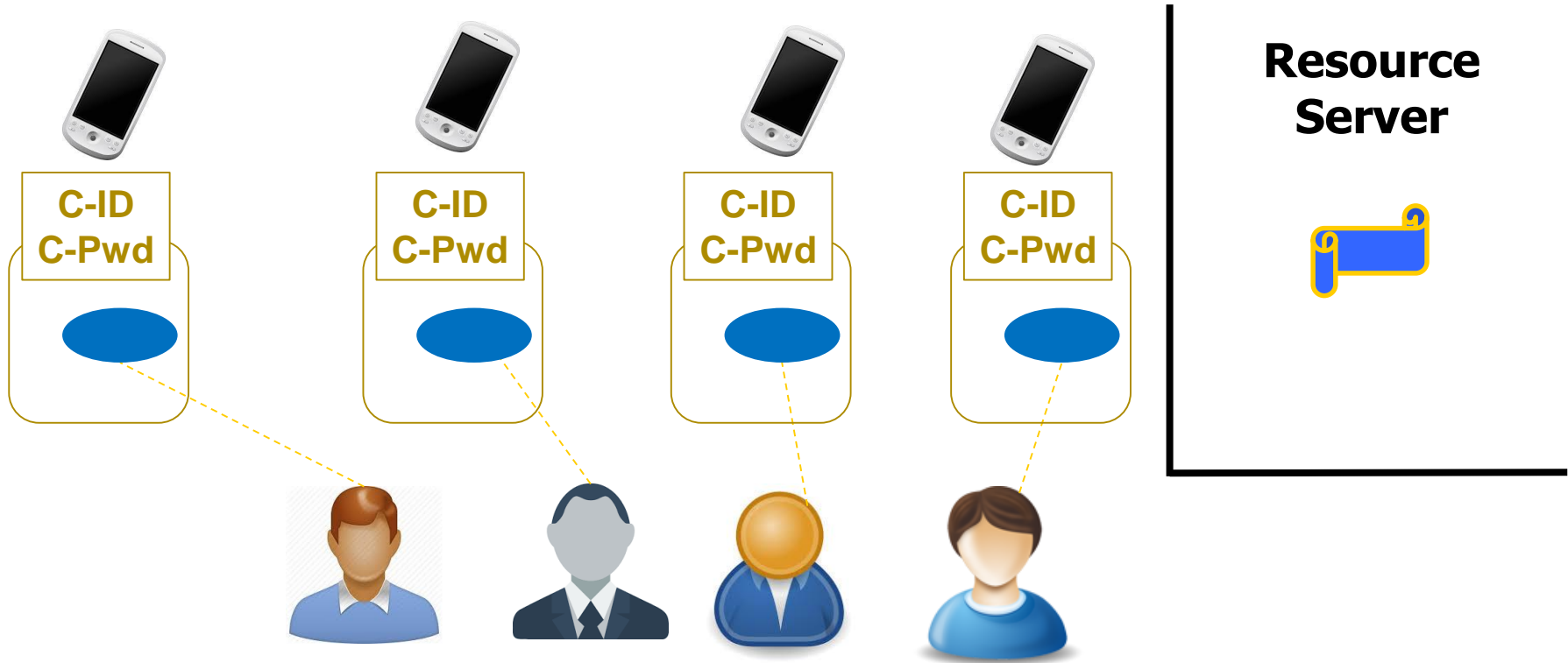
- ❑ Client Program:
 - ❑ Program downloaded from a **public** store and executed on **PC**
 - ❑ Dropbox / Google Drive / Evernote
 - ❑ App downloaded from a **public** store and executed on **Smartphone**
 - ❑ Facebook / Twitter / Unicredit / Postepay / ...
- ❑ Problems and solutions for ClientPWD / AuthGrant
- ❑ We will not analyze the other flows in detail

Client = Webapp

- ❑ One program
- ❑ Many refresh tokens (one for each RO)

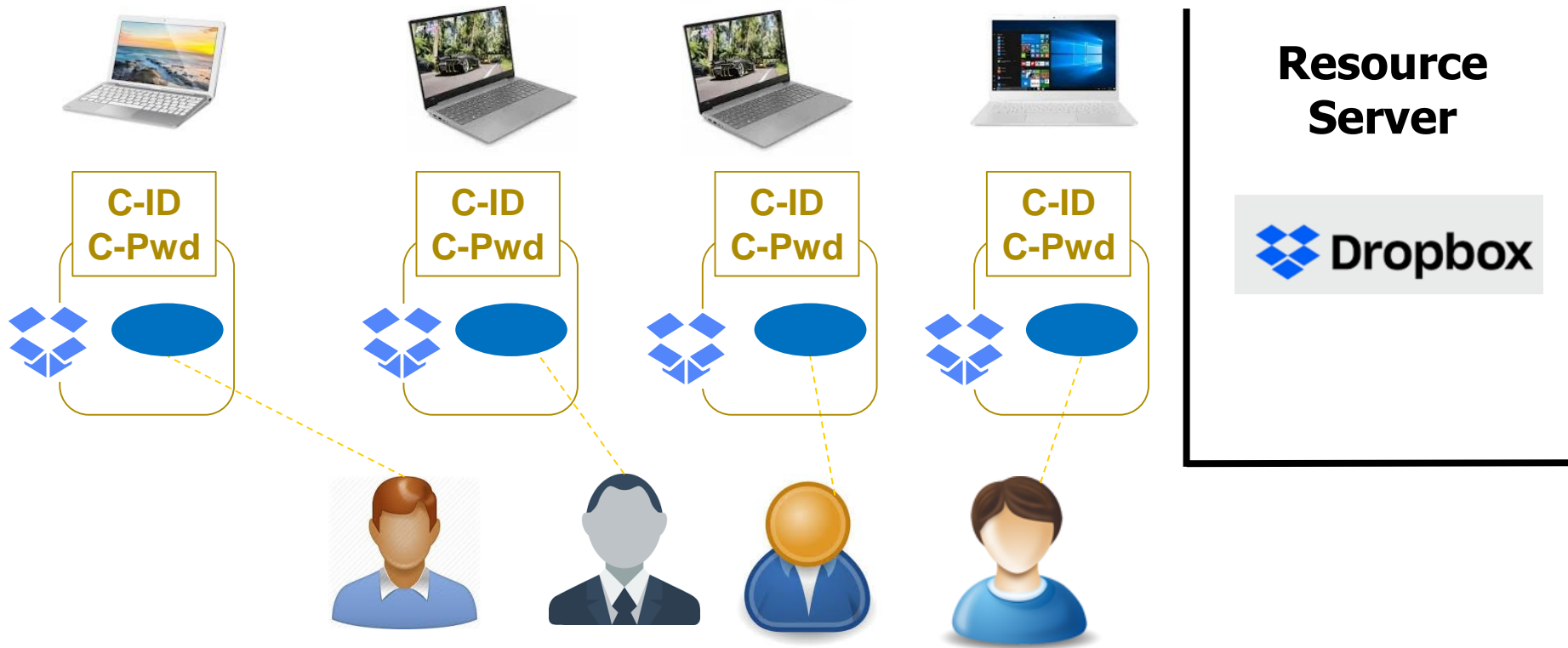


Client = Public App on Smartphone



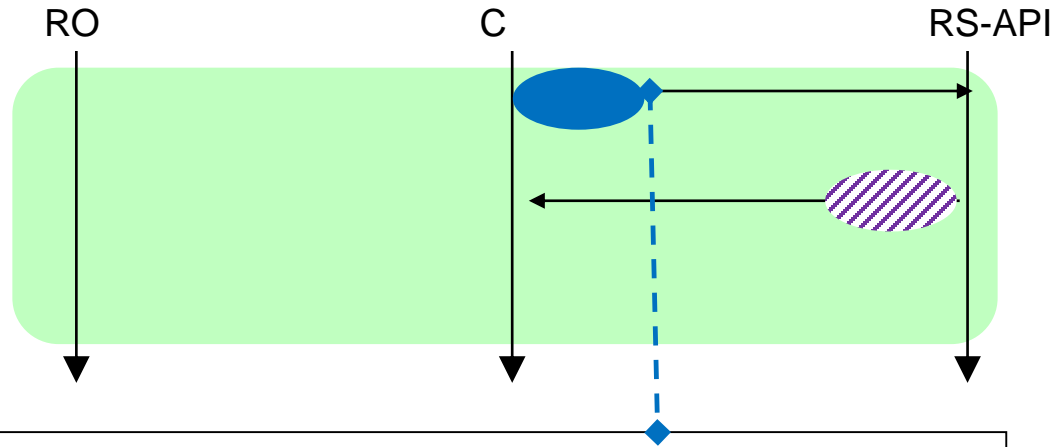
- ❑ One **copy** of the **same** program
- ❑ One refresh token (for the **respective** RO)

Client = Public App on PC



- ❑ One **copy** of the **same** program
- ❑ One refresh token (for the **respective** RO)

Refresh token → Access token



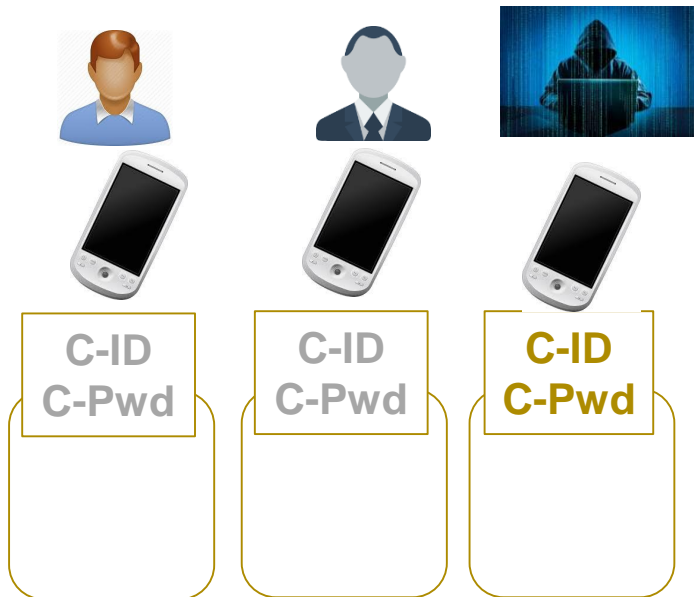
ClientID
ClientPassword

```
POST ...
Host: ...
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=authorization_code&
  code=Sp1xl0BeZQQYbYS6WxSbIA&
  redirect_uri=ClientRedirection
```

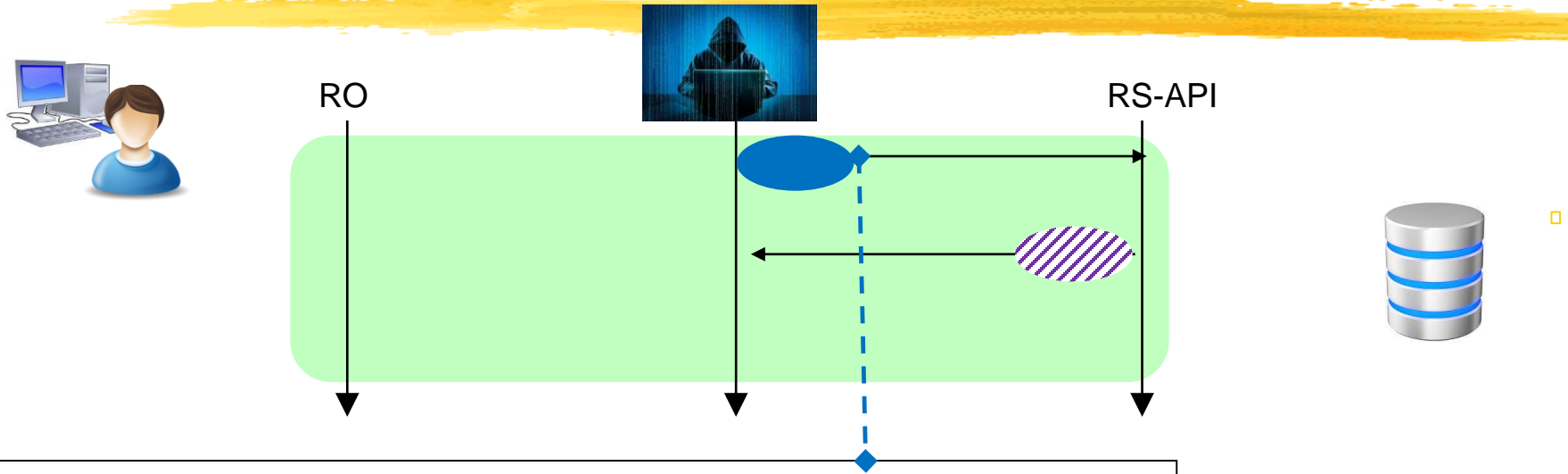
- ❑ Credentials of C
- ❑ **NOT** of the Resource Owner

Problem?



- ❑ C is the **same app** for **all the users**
- ❑ Attacker may:
 - ❑ Reverse engineering the app
 - ❑ **Extract the credentials of C**
- ❑ Also apps on PC

Refresh → Access: No problem



```
POST ...
Host: ...
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&
  refresh_token=????&
  redirect_uri=ClientRedirectionURL
```

ClientID
ClientPassword
Reverse engineered

Realistic Threat Model: "Lookalike" + ClientCreds

□ Attacker actions:

1. Develop **EvilClientApp** with the **same look** as **GoodClientApp**
⇒ Can impersonate **GoodClientApp** to Users
2. Try to convince U to install and delegate **EvilClientApp**
⇒ Obtain a **legitimate** Grant(**GoodClientApp**, U@RS)
⇒ Convert to RefreshToken(**GoodClientApp**, U@RS)

Reverse engineered
GoodClientApp credentials

- ## □ User believes to install and delegate **GoodClientApp** (but installs and delegates **EvilClientApp**)



Defense: PKCE Flow

- ❑ Clients that **cannot store credentials securely**:
 - ❑ Every **installation** has a **different ClientPassword**
 - ❑ Flow for converting Grant→ Refresh different (out of scope)
- ❑ Installation-bound credentials may be obtained either:
 1. during the **download** process from the application market, or
 2. during **installation** on the device
- ❑ **Automated** mechanism currently **not defined by OAuth**.
- ❑ Used by apps on PC (Google Drive, Dropbox, etc)
- ❑ More details:
 - ❑ <https://tools.ietf.org/html/rfc6819#section-5.2.3.4>
 - ❑ search "OAuth for Mobile and Desktop Apps"

Realistic Threat Model: SOLVED

□ Attacker actions:

1. Develop EvilClientApp with the **same look** as GoodClientApp
⇒ Can impersonate GoodClientApp to Users
2. Try to convince U to install and delegate EvilClientApp
⇒ Obtain a **legitimate** Grant(**GoodClientApp**, U@RS)
⇒ Convert to RefreshToken(**GoodClientApp**, U@RS)

~~Reverse engineered
GoodClientApp credentials~~



OAuth: Trust and Security issues



Realistic Threat Model: "Lookalike" and nothing else

□ Attacker actions:

1. Develop EvilClientApp with the **same look** as GoodClientApp
⇒ Can impersonate GoodClientApp to Users
2. Register **EvilClientApp** with RS
⇒ Obtain a **legitimate** Client-ID+Client-PWD
3. Try to convince U to install and delegate EvilClientApp
⇒ Obtain a **legitimate** Grant(**EvilClientApp**, U@RS)
⇒ Convert to RefreshToken(**EvilClientApp**, U@RS)

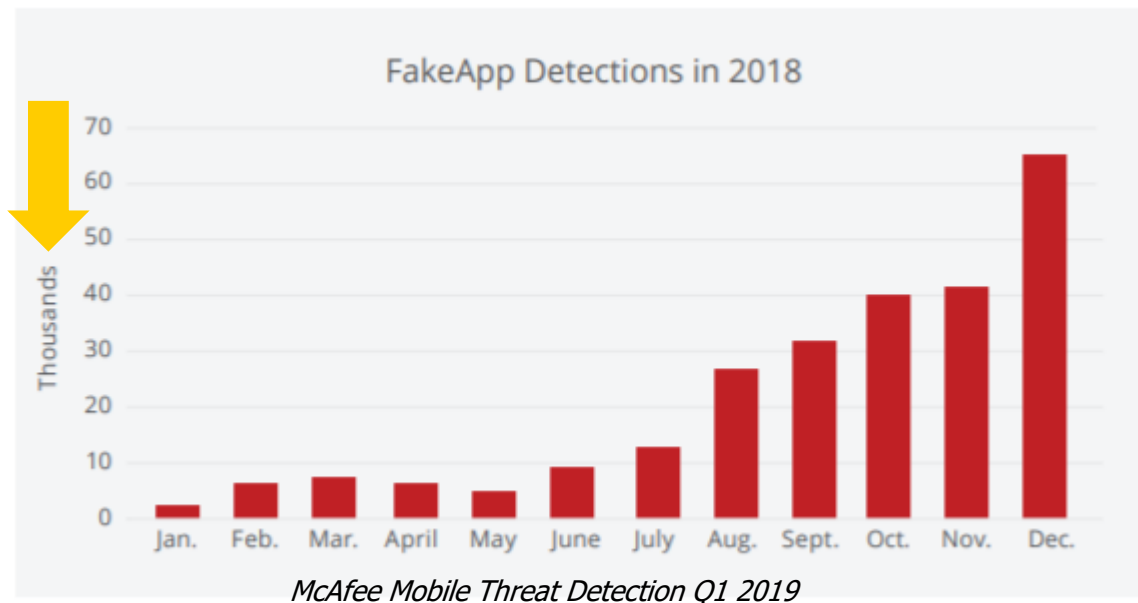
Legitimate EvilClientApp
credentials



- ## □ User believes to install and delegate GoodClientApp (but installs and delegates EvilClientApp)

"App phishing"

Many examples of **fake apps**
(**EvilClient** looks like **GoodClient**)



Hmmm...

- ❑ Threat Model **Network Attacker**
- ❑ Cannot **alter any code / steal any information** on RO, C, RS
- ❑ This attack is included in this threat model

*OAuth does **not** work?*



OAuth solves a **DIFFERENT** problem



- OAuth:

- **Secure transfer** of authorization **assertions** over **untrusted channels**

- This attack:

- RO **believes** to have authorized C to do something on RS
 - RO has **actually** authorized a **different** C'

Realistic Threat Model: Lie

□ Attacker actions:

1. Develop **EvilClientApp**
2. Register **EvilClientApp** with RS
3. Try to convince users to install and delegate **EvilClientApp**

□ **EvilClientApp**:

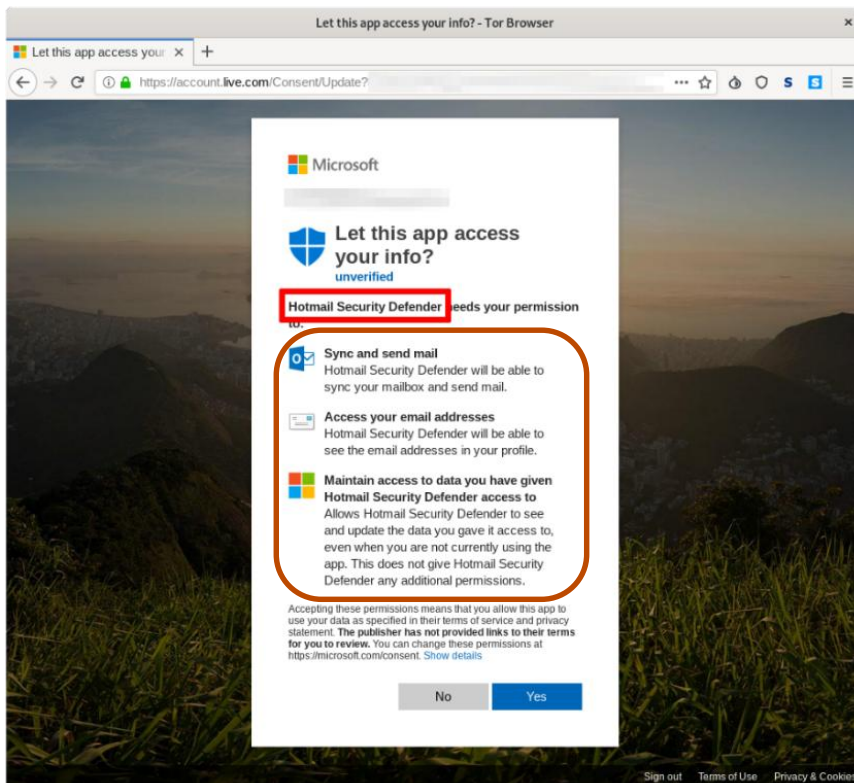
*I will use Access Right A on resources R for doing **X***

□ ...but **EvilClientApp** actually does **Y**



Example

- ❑ **C** = Hotmail Security Defender
- ❑ **RS** = Microsoft Cloud



- ❑ How C will use the read data?
- ❑ How C will update those data?
- ❑ Can OAuth give you any guarantee in this respect?

Misplaced Trust



- ❑ OAuth:

- ❑ **Secure transfer** of authorization **assertions**
over **untrusted channels**

- ❑ RO **trusts** C

- ❑ OAuth allows conveying this trust securely

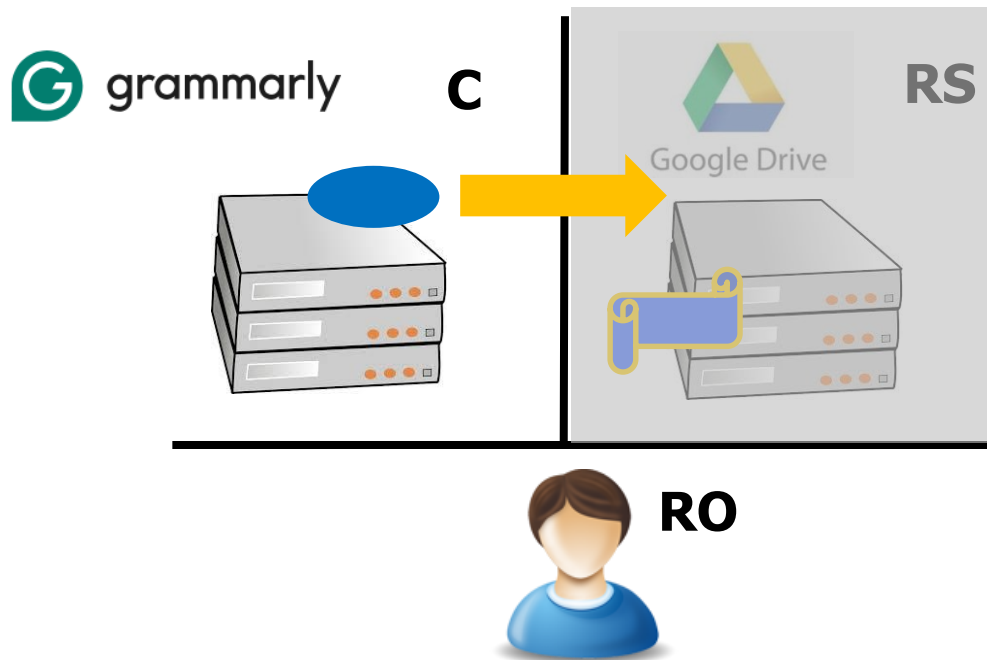
- ❑ **Trust** might turn out to be **unjustified**

- ❑ Subtle but extremely important

Understanding Trust in OAuth

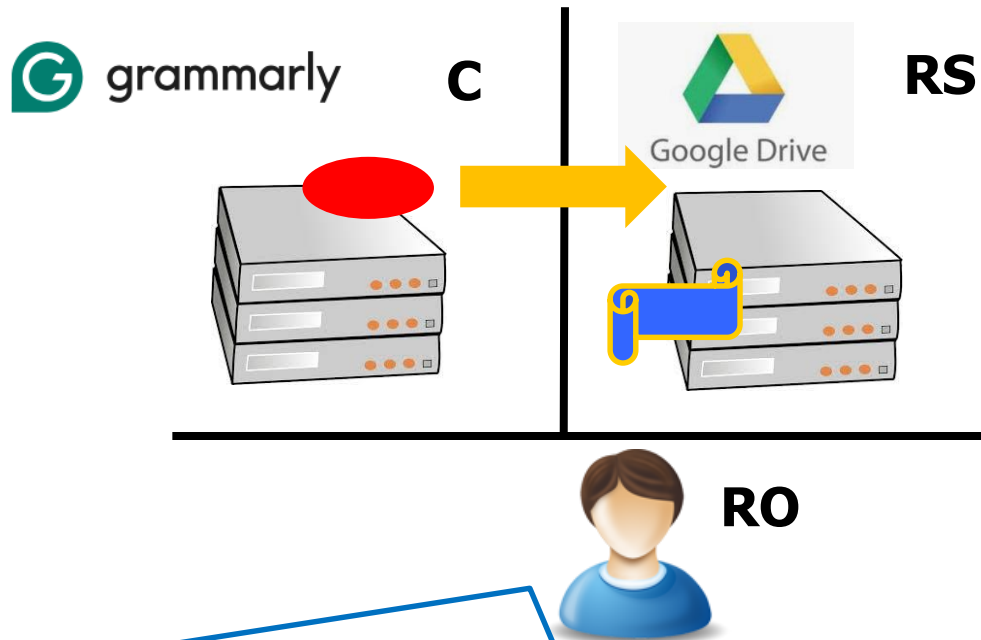


RO trusts C (I)



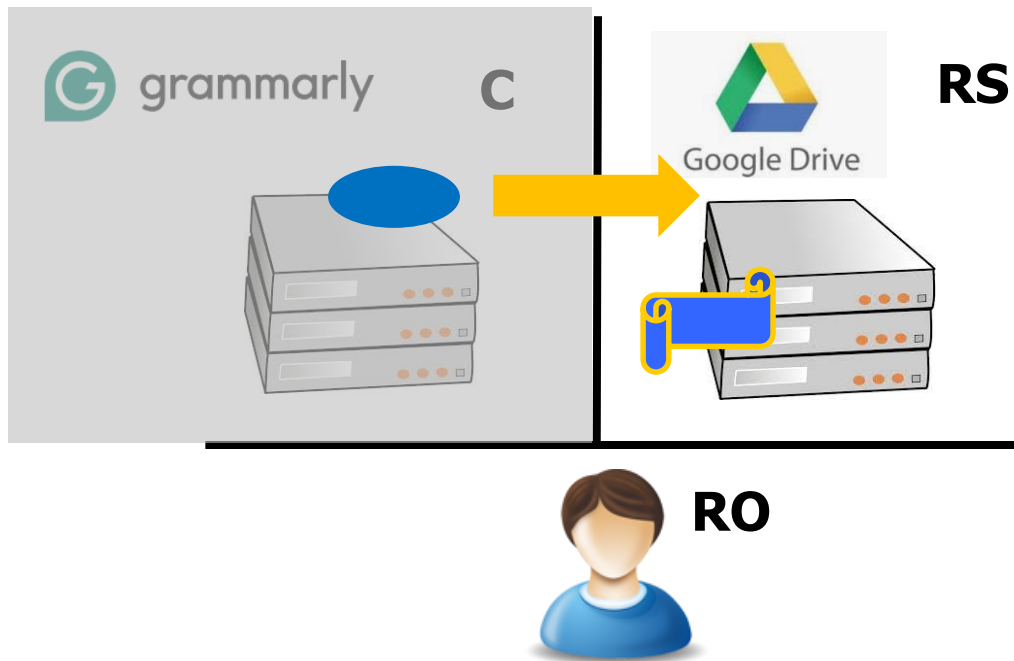
- ❑ C will store $\text{Token}(C, \text{RO}@RS)$ **securely**
- ❑ C will **not** have any OAuth **vulnerability**
(otherwise some other EvilC might obtain an equivalent token)

RO trusts C (II)



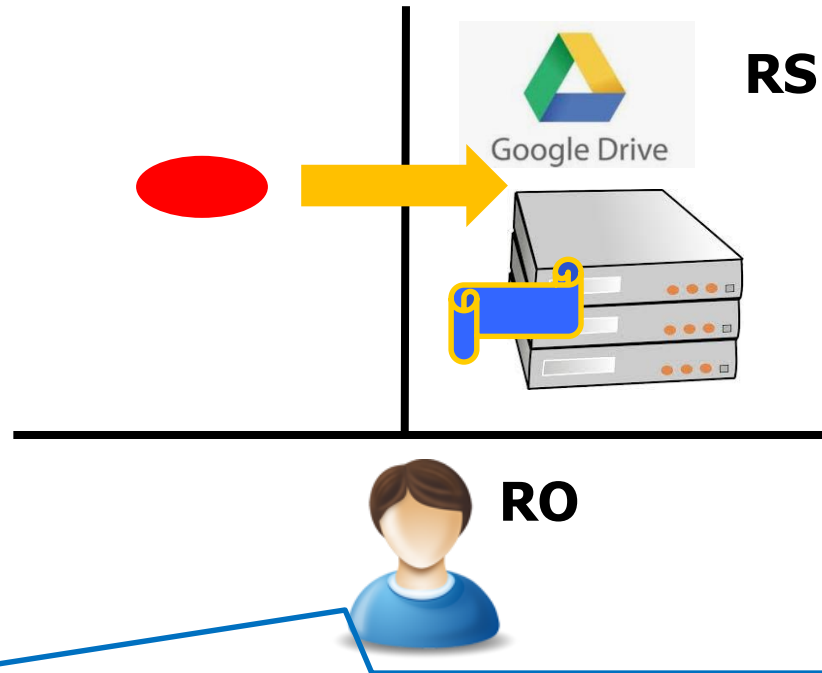
- ❑ Will C store "my token for Google Drive" **securely**?
- ❑ Will C implement OAuth so that "my token for Google Drive" **cannot be cloned**?

RO trusts RS (I)



- ❑ RS will allow accessing RO resources **only** with Token(C, RO@RS)
- ❑ RS will **not** have any OAuth **vulnerability**
(otherwise some other EvilC might obtain an equivalent token)

RO trusts RS (II)



- ❑ Will Google Drive allow accessing my resources **only** with a **Token that I decided to grant**?
- ❑ Will Google Drive implement OAuth so that no **Token that I decided to grant cannot be cloned**?

Stealing Tokens

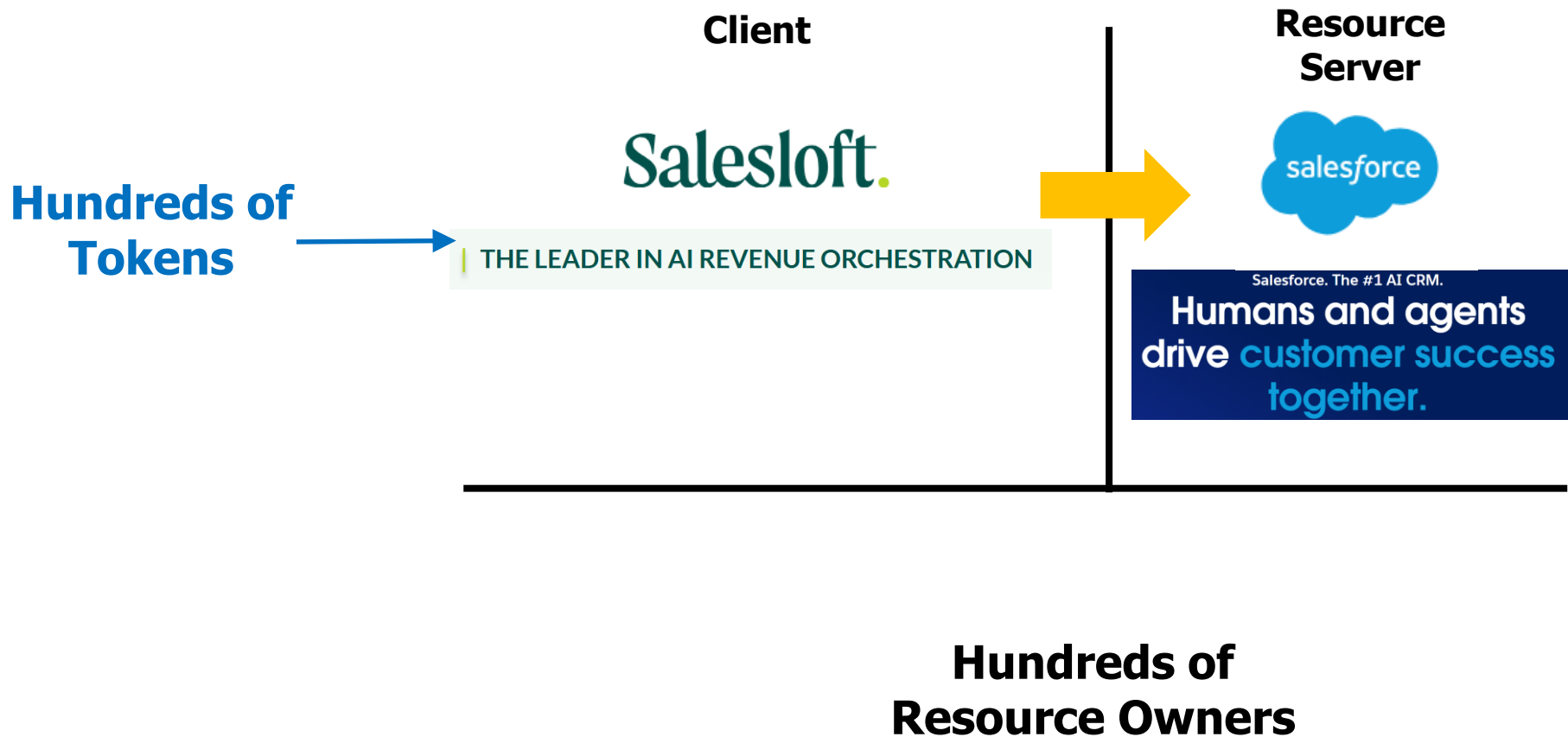


Threat Model: Realistic?

- ❑ Threat Model **Network Attacker**
- ❑ Cannot **alter any code** / **steal any information** on RO, C, RS



AI Chat Agent → CRM



August-September 2025 (I)



BLEEPINGCOMPUTER

Salesloft breached to steal OAuth tokens for Salesforce data-theft attacks

August 26, 2025

ShinyHunters claims 1.5 billion Salesforce records stolen in Drift hacks

September 17, 2025

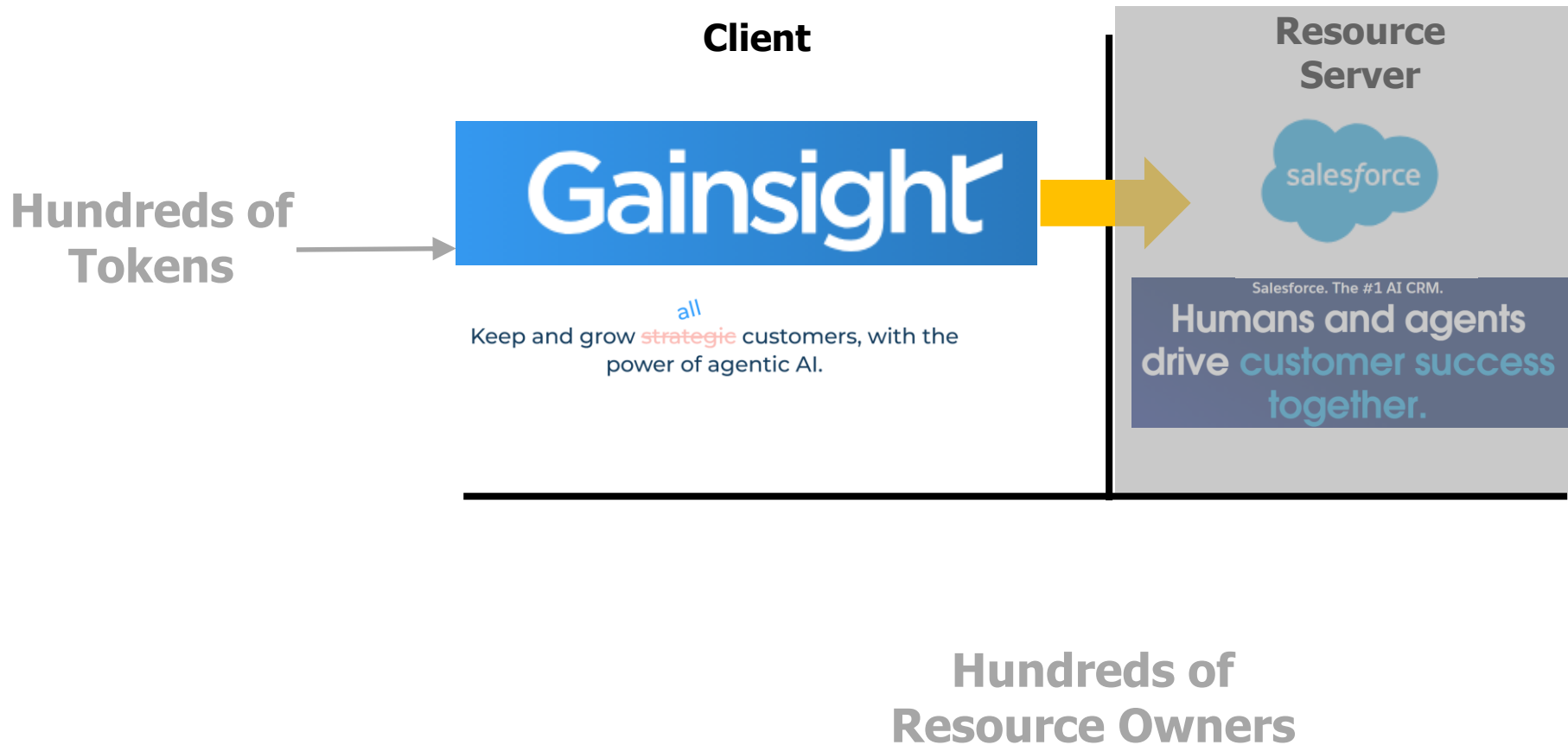
August-September 2025 (II)

Salesloft Drift Breach - Track the Salesforce Incident

Companies affected by the widespread Salesloft Drift OAuth token compromise that targeted Salesforce customer instances

- ❑ Using the stolen OAuth credentials, the threat actor **bypassed normal authentication (including MFA)** and **exfiltrated large volumes of Salesforce data** from **hundreds of organizations**.
- ❑ The attackers also took steps to cover their tracks by **deleting Salesforce query job records** after data exports.
- ❑ The activity focused on finding credentials within the exfiltrated Salesforce data, specifically AWS access keys, passwords, and Snowflake tokens

November 2025 (I)



November 2025 (II)



Salesforce-linked data breach claims 200+ victims, has ShinyHunters' fingerprints all over it

The Register®

Thu 20 Nov 2025

Salesforce has disclosed another third-party breach in which criminals - likely ShinyHunters (again) - may have accessed hundreds of its customers' data.

This time, the suspicious activity involves Gainsight-published applications connected to Salesforce, which are installed and managed directly by customers.

What happened



- ❑ March - June 2025: the threat actor accessed the **Salesloft GitHub account**. With this access, the threat actor was able to:
 - ❑ download content from multiple repositories
 - ❑ add a guest user
 - ❑ establish workflows.

- ❑ The threat actor then:
 - ❑ Accessed Drift's **AWS** environment
 - ❑ **Obtained OAuth tokens** for Drift customers' technology integrations.
 - ❑ **Used the stolen OAuth tokens** to access data via Drift integrations.

What happened: MITRE ATT&CK (I)

Privilege Escalation

14 techniques

Account Manipulation (7)	Additional Cloud Credentials
	Additional Email Delegate Permissions
	Additional Cloud Roles
	SSH Authorized Keys

Defense Evasion

47 techniques

Credential Access

17 techniques

Lateral Movement

9 techniques

Steal Application Access Token

Use Alternate Authentication Material (4)	Application Access Token
	Pass the Hash
	Pass the Ticket
	Web Session Cookie

Use Alternate Authentication Material (4)	Application Access Token
	Pass the Hash
	Pass the Ticket
	Web Session Cookie

What happened: MITRE ATT&CK (II)

Persistence

23 techniques

Cloud
Application
Integration

Additional Cloud Credentials
Additional Email Delegate Permissions
Additional Cloud Roles
SSH Authorized Keys

Account
Manipulation (7)

Credential Access

17 techniques

Steal
Application
Access Token

GitHub = OAuth Resource Server (I)



- ❑ My apps/Projects
 - ❑ Delegated to operate on **my** GitHub repositories
 - ❑ **I** have to store tokens securely



GitHub = OAuth Resource Server (II)

