

# AUTHENTIFIZIERUNG UND AUTORISIERUNG IM WEB

Verteilte Systeme

Prof. Dr. Georg Hinkel  
28.06.2024

# GLIEDERUNG

Datum	Vorlesung	Übungsblatt	Abgabe
19.04.2024	Einführung	HamsterLib	06.05.2024
26.04.2024	Netzwerkprogrammierung	Theorie	
03.05.2024	World Wide Web	HamsterRPC 1	20.05.2024
10.05.2024	Remote Procedure Calls	Theorie	
17.05.2024	Webservices	HamsterRPC 2	03.06.2024
24.05.2024	Fehlertolerante Systeme	Theorie	
31.05.2024	Transportsicherheit	HamsterREST	17.06.2024
07.06.2024	Architekturen für Verteilte Systeme	Theorie	
14.06.2024	Internet der Dinge	HamsterIoT	01.07.2024
21.06.2024	Namen- und Verzeichnisdienste	Theorie	
28.06.2024	Authentifikation im Web	HamsterAuth	15.07.2024
05.07.2024	Infrastruktur für Verteilte Systeme	Theorie	
12.07.2024	Wrap-Up	HamsterCluster (Bonus)	16.08.2024

## Agenda

- Authentifizierungstypen
- OAuth 2.0
- SAML
- Open ID Connect

## Lernziele

- Vorgehensweise moderner Authentifizierung erklären können
- Für Anwendung geeigneten Flow auswählen können

- Identifikation
    - Feststellen der Identität
    - Beispiel: Eingabe Username
  - Authentifikation
    - Sicherstellen der Identität
    - Beispiel: Eingabe Passwort
  - Autorisierung
    - Erteilen von Zugriffsrechten
    - Beispiel: Aufgrund Rollen oder Mitgliedschaften
- Typische Abhängigkeiten
    - Authentifikation benötigt Identifikation
    - Autorisierung benötigt Authentifikation
  - Abhängigkeiten sind nicht zwangsläufig
    - Identifikation ohne Authentifikation
      - Gesichtserkennung
    - Autorisierung ohne Authentifikation
      - Wohnungsschlüssel

# WARUM NEHMEN WIR NICHT EINFACH DAS LDAP?



- **Wissen**
  - Passwort
  - PIN
  - ...
- **Besitz**
  - Zertifikate
  - Token
  - ...
- **Biometrie**
  - Fingerabdruck
  - Gesichtserkennung
  - Iris-Scan
  - ...



# BEISPIEL SOFTWARE TOKEN: OTP

Time-based One-Time-Passwords, RFC 6238

- Nutzerspezifisches Geheimnis, typischerweise per QR-Code an Smartphone übertragen
  - Sichere Aufbewahrung des Schlüssels, bspw. durch TPM-Modul
- HMAC-SHA1 Algorithmus, um aus Schlüssel und Systemzeitscheibe Zifferncode abzuleiten
  - HMAC ist kryptografische Einwegfunktion mit symmetrischem Schlüssel
  - Zifferncode typischerweise 6-stellige Zahl
  - Zeitscheiben üblicherweise je 30s ab Unix-Epoche (1.1.1970)
- Prüfe, ob Zahlencode identisch auf Client und Server
  - Ggf. Code für angrenzende Zeitscheiben auch zulässig



- Unternehmen gehen dazu über, Authentifizierungen nur mit Passwörtern nicht mehr zu akzeptieren
  - Mehrere Faktoren zur Authentifizierung verpflichtend (MFA)
  - Beispiel: 2 Faktoren (2FA)
- Teils Authentifizierungsschemen ohne Passwörter
  - Beispiel: Push-Nachrichten (Microsoft)
- Beispiele
  - US National Information Assurance Glossary: mindestens 2FA
  - Europäische Zentralbank
    - Mindestens 2FA
    - Verwendete Faktoren gegenseitig unabhängig, mindestens ein Faktor "non-reusable and non-replicable"



- „Forms Authentication“
  - Individuelle Accounts für jede Website
  - Anmeldeinformation wird in Cookie, Request-Parameter oder Session gespeichert
  - Jede Anwendung hat eigene Nutzerregistrierung, Passwort-Policy, Nutzeradministration, ...
    - Gute Frameworks, um diese Aufgaben zu übernehmen
  - Kein zentrales Nutzerverzeichnis (LDAP)
- Nachteile
  - Jede Website benötigt eigenes Passwort
  - Menschen können sich nicht beliebig viele Passwörter merken
    - Wiederverwendung von Passwörtern
    - „Gedächtnisstützen“ am Bildschirmrand
  - Sicherheitsniveau der Aufbewahrung ist für Nutzer intransparent

# WAS BRAUCHT ES, UM FORMS-AUTHENTIFIZIERUNG RICHTIG HINZUBEKOMMEN?

- Strategie zur Passwortspeicherung
  - Speichere Hashwerte, keine Passwörter
  - Individuelle Salts pro User, teilweise zusätzlich Pfeffer
  - Empfohlene Algorithmen für Passwort-Hashing ändern sich von Zeit zu Zeit (aktuell Argon2id)
- Email Verifikation
- Nutzerverwaltung
  - Administration für alle Bereiche der Anwendung
- Single-sign-on
- 2FA
- Biometrie (WebAuthN)
- ...

Latentes Risiko des  
Datendiebstahls übernehmen?

# DELEGIERTE AUTORISIERUNG

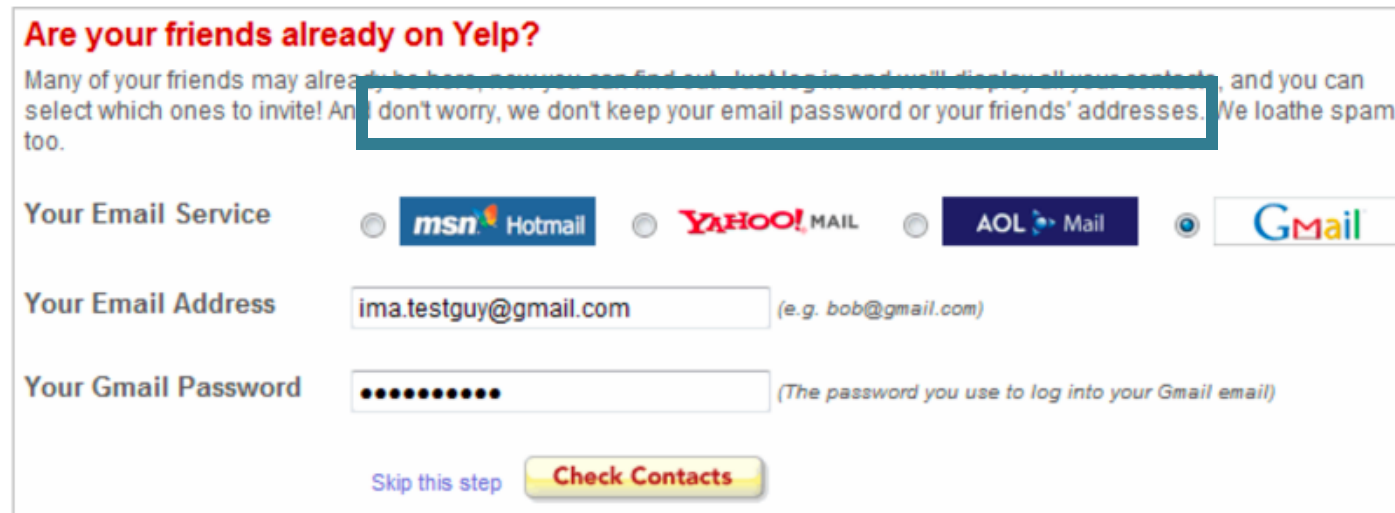
## Einführung

- Idee ursprünglich: Authentifizierung im Namen des Nutzers
  - Analogie Haustürschlüssel
  - ...ermöglicht nicht Zugriff auf Bankkonto...
- Beispiel: Online-Druckerei braucht Leserechte auf Bilder, die in Cloud-Dienst gespeichert sind
  - Online-Druckerei sollte keine E-Mails lesen dürfen, die beim gleichen Provider gespeichert sind
- Problem: Wie setzen wir das um?

# DELEGIERTE AUTORISIERUNG

## Naiver Ansatz

- Idee: Frage Nutzer einfach nach den Zugangsdaten
  - Tatsächlich in den Anfangstagen des Web verwendet
- Problem: Unklar, was der Dienst damit sonst noch alles macht



**Are your friends already on Yelp?**

Many of your friends may already be on Yelp, so we can find out, but we won't display all your contacts, and you can select which ones to invite! And don't worry, we don't keep your email password or your friends' addresses. We loathe spam, too.

Your Email Service ☐ msn Hotmail ☐ YAHOO! MAIL ☐ AOL Mail ☒ Gmail

Your Email Address  (e.g. bob@gmail.com)

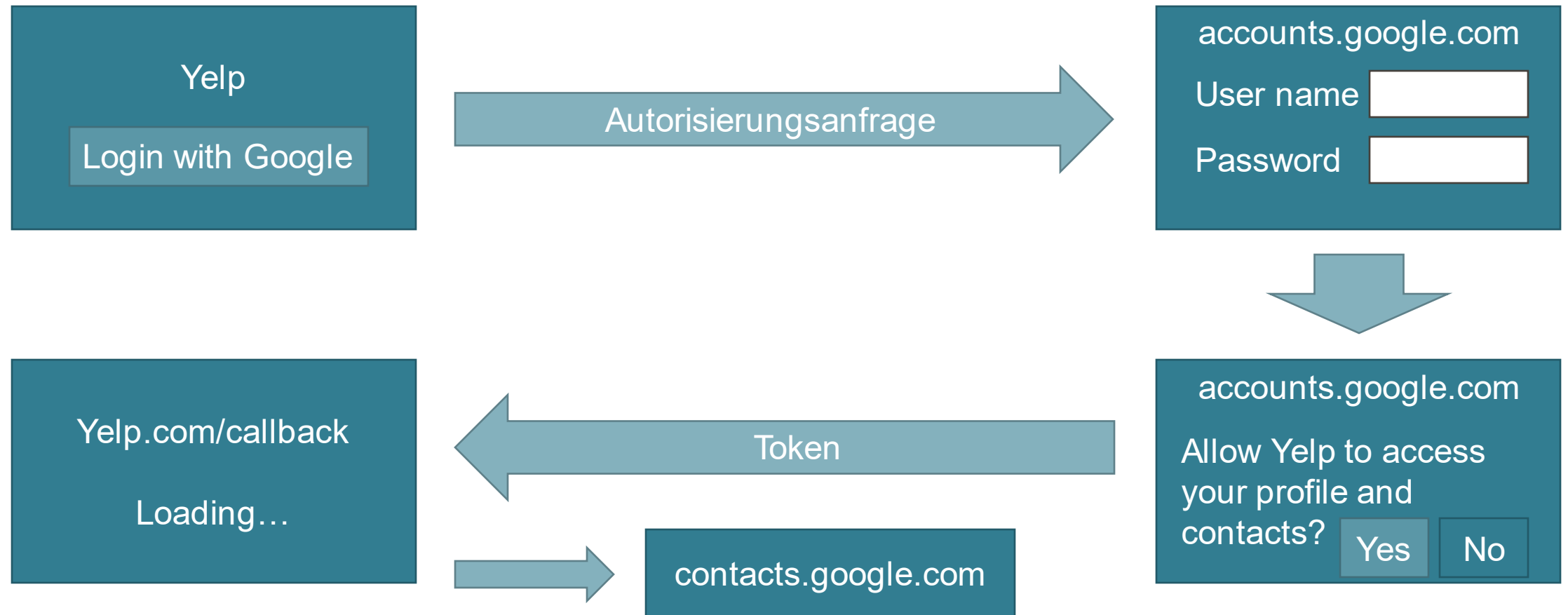
Your Gmail Password  (The password you use to log into your Gmail email)

[Skip this step](#) [Check Contacts](#)

[Bild: Coding Horror]

# DELEGIERTE AUTHENTIFIZIERUNG

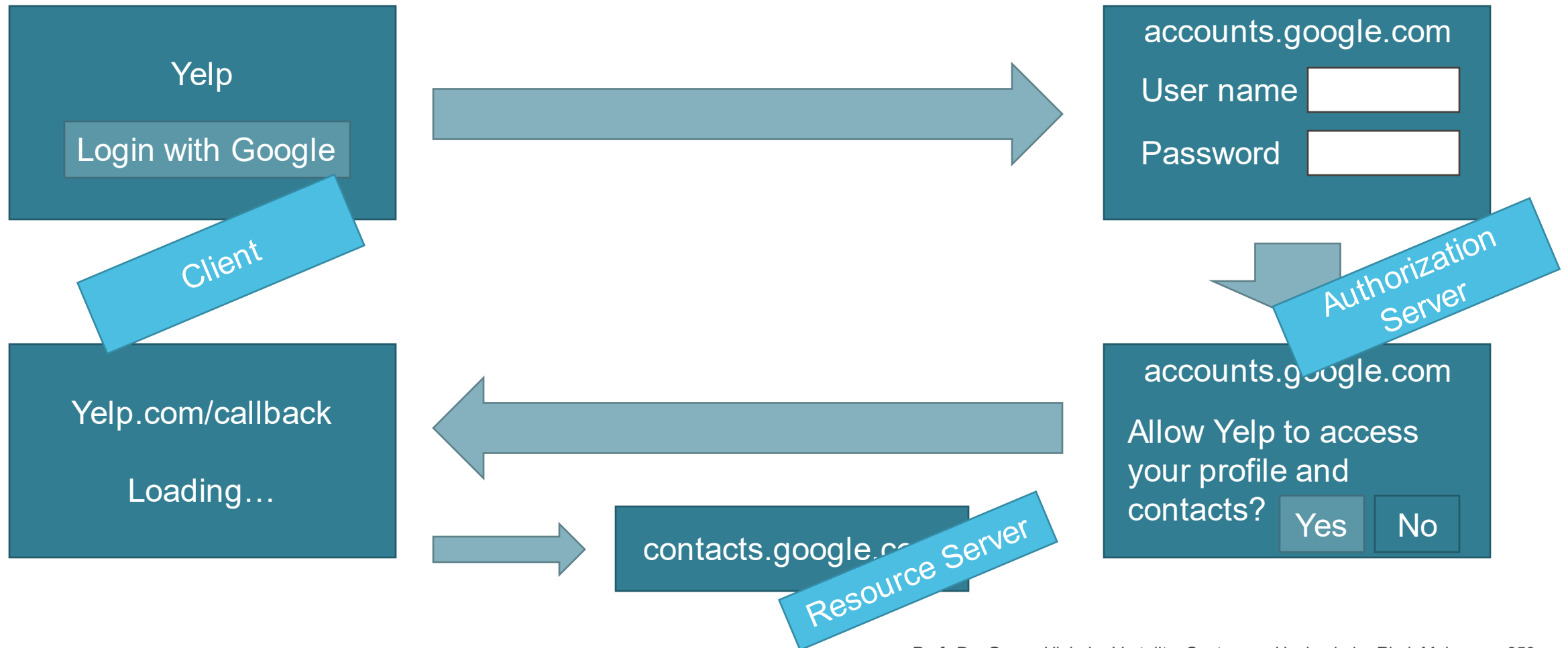
OAuth 2.0



- Resource Owner
  - Partei, die Rechte an den angefragten Ressourcen hat (Nutzer)
- Client
  - Partei, die Rechte anfragt, typischerweise Webanwendung (im Beispiel: Yelp)
- Authorization Server
  - Server, der Zugriff auf Ressourcen verwaltet (im Beispiel `accounts.google.com`)
- Resource Server
  - Server, der die angefragte Ressource bereitstellen kann (im Beispiel `contacts.google.com`)

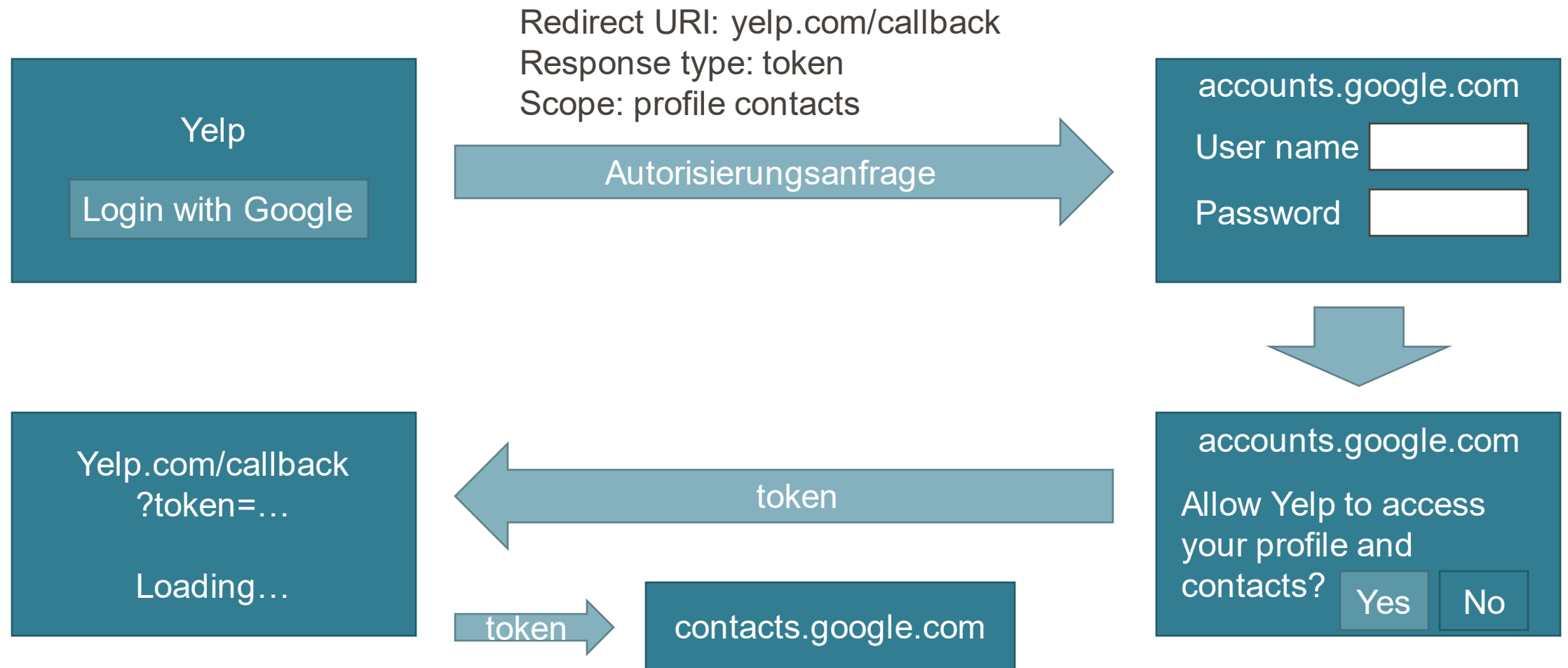
# OAUTH 2.0

## Terminologie



# OAUTH 2.0

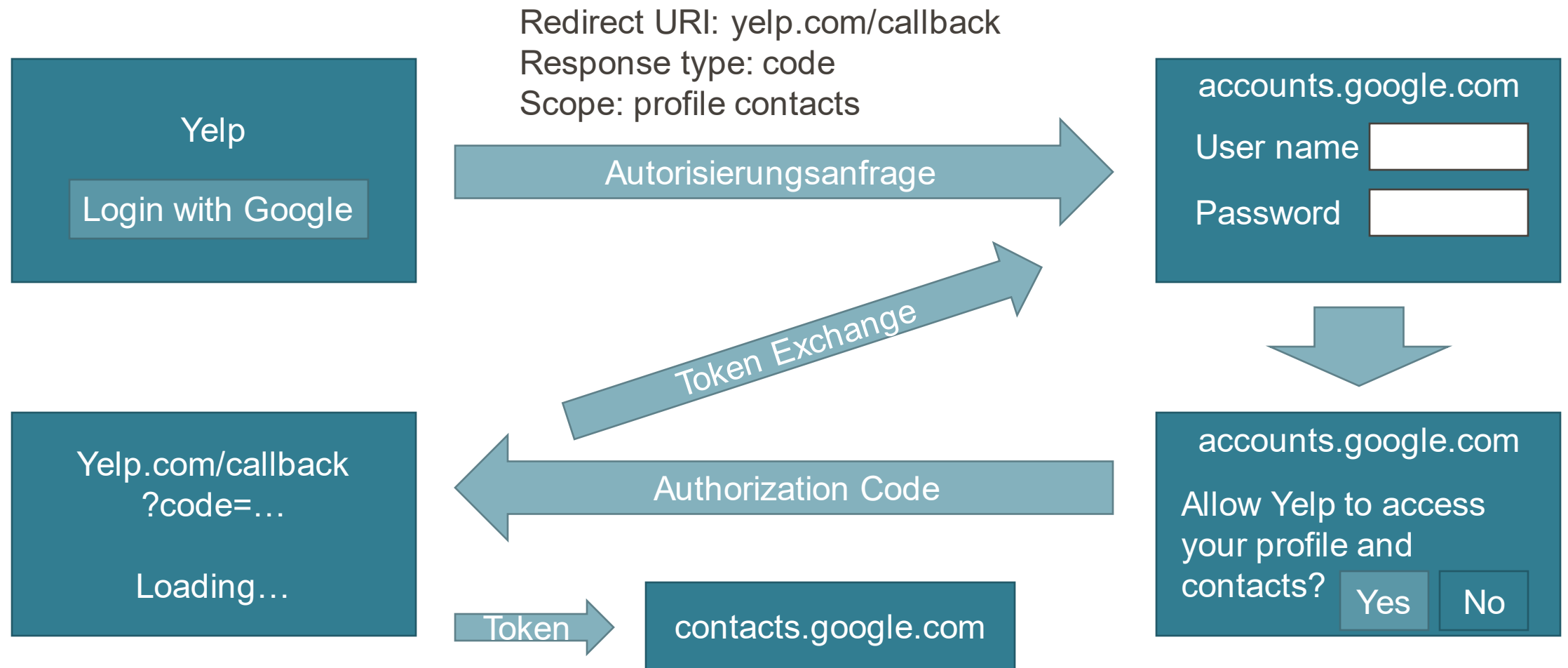
## Implicit Flow



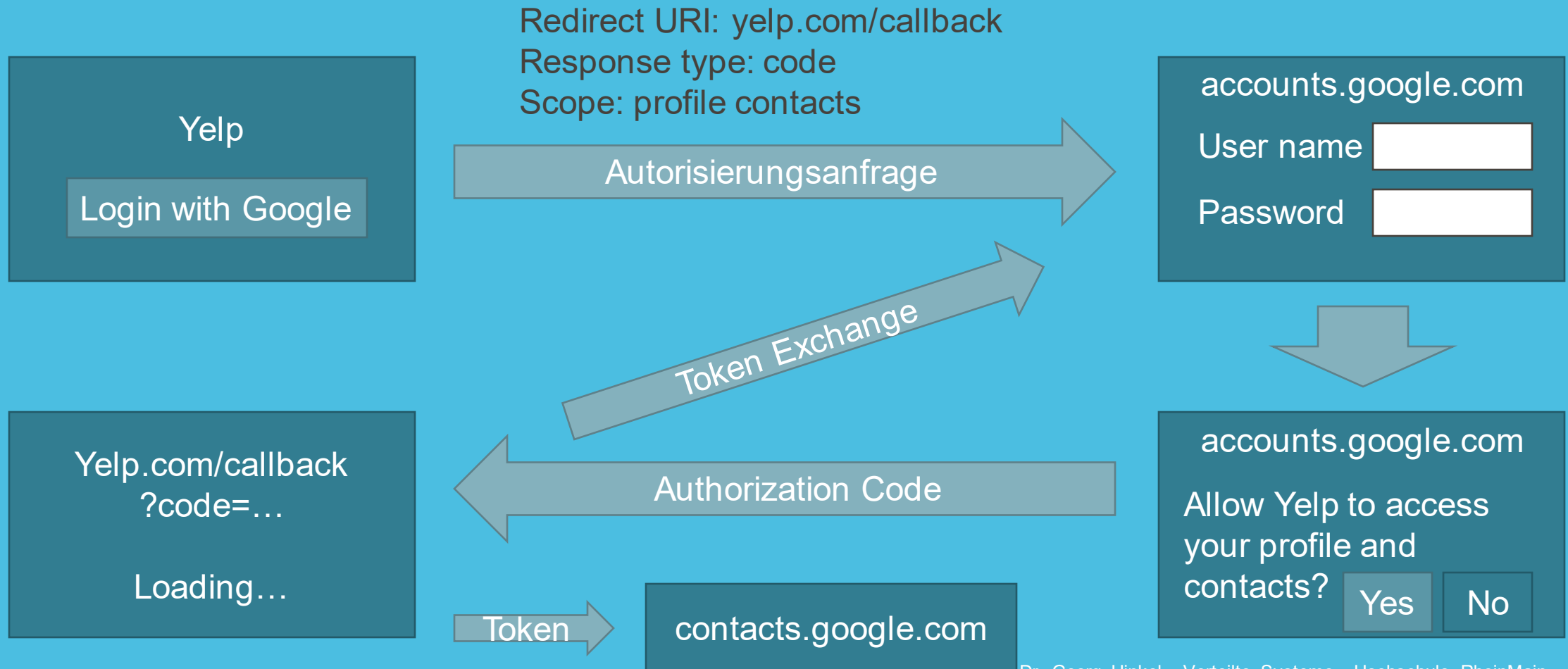


# OAUTH 2.0

## Authorization Code Flow



# WARUM CODE FLOW?



# WARUM CODE FLOW?

- Client ist typischerweise Browser
  - Keine Kontrolle über Ausführungsumgebung
    - Computer könnte mit Keyloggern, Trojanern, ... infiziert sein
  - Keine Kontrolle über Code-Ausführung
    - Nutzer könnte Entwicklertools verwenden, ...
- Ziel: Unbefugte sollten Token nicht erlangen können
  - Token wird nur an befugte Parteien ausgegeben
  - Token hat nur kurze Gültigkeit (typischer Wert: 10min)
    - Client darf innerhalb der 10min Token „verlängern“ (neuer Token mit längerer Haltbarkeit)
  - Idealerweise gesichert durch Client Secret (außer SPA)
- Außerdem: Behebt Längenbegrenzungen
  - URLs haben begrenzte Länge (RFC 7230:  $\geq 8000$  Zeichen, Praxis: 2000 Zeichen)
  - In SPAs: Cross-Origin Resource Sharing

- Problem: Was ist ein Scope?
- Generische Antwort: Mittel, um Zugriffsbereich eines Tokens einzugrenzen
  - Analogie: Nicht alle Prozesse mit Admin-Rechten laufen lassen, auch wenn man Admin ist
- Auffassung sehr unterschiedlich
  - Anwendungsfälle einer Anwendung → ermöglicht granulare Zugriffssteuerung
  - Komplette Anwendungen (e.g. E-Mails, Kalender, Kontakte, Dateien)
  - Gemisch aus beiden Ansätzen

# OAUTH 2.0 TERMINOLOGY (CONTINUED)

- Redirect URI
  - URI, an den der Authorization Server nach Authentifizierung einen Redirect stellt
- Access Token
  - Token, der zum Zugriff auf die angefragte Ressource berechtigt
- Audience
  - Server, für den ein Token ausgestellt worden ist
- Access Grant
  - Code, der beim Authorization Server gegen einen Access Token ausgetauscht werden kann
- Refresh Token
  - Token, der die Gültigkeit des Access Token verlängert

- Resource Owner Password Flow
  - Request enthält direkt Nutzernamen und Passwort
  - Verwendet vorrangig zur Unterstützung bestehender Legacy-Anwendungen
- Client Credentials Flow
  - Client authentifiziert sich mit Client-spezifischem Passwort
  - Verwendet für Server-zu-Server Kommunikation im Hintergrund (Daemon)
- Device Authorization Grant Flow
  - Authentifizierung von Geräten mit eingeschränkter Nutzereingabe (e.g. Smart-TV)
  - Authentifizierung durch Browser in separatem Gerät
- On-Behalf-of-Flow
  - Authentifizierung von Diensten im Namen des Nutzers
  - Audience/Scope von bestehendem Access Token wird geändert

# DELEGIERTE AUTHENTIFIZIERUNG

## Pseudo-Authentifizierung mit OAuth 2.0

- Idee: Verwende OAuth 2.0 für delegierte Authentifizierung
- Hauptproblem: Token enthält keine Nutzerdaten (~ Analogie: Haustürschlüssel)
  - Name
  - Email-Adresse
- Endpunkt, um Nutzerdaten zu bekommen nicht standardisiert
  - Analogie: Welches Haus macht der Schlüssel denn jetzt auf und was genau bringt einem das?
  - Zusätzlicher Aufwand, um Nutzerdaten abzufragen

# DELEGIERTE AUTHENTIFIZIERUNG

## Überblick

Protokoll	SAML 2.0	Open ID Connect 1.0
Entstehung	2005	2014
Standardisierungsgremium	OASIS	OpenID Foundation
Zweck	Single-Sign-On	Delegierte Authentifizierung auf Basis von OAuth 2.0
Token-Format	XML	JSON (JWT), wie OAuth 2.0
Bezeichnung der Token	Assertion	Claim
Bezeichnung der Anwendung	Service Provider	Relying Party

- Beide Protokolle stark verbreitet (oft simultan)
  - SAML wahrscheinlich weiter verbreitet, aber OIDC zunehmend populär
  - OIDC häufig einfacher zu implementieren
- Im Folgenden Fokus auf Open ID Connect



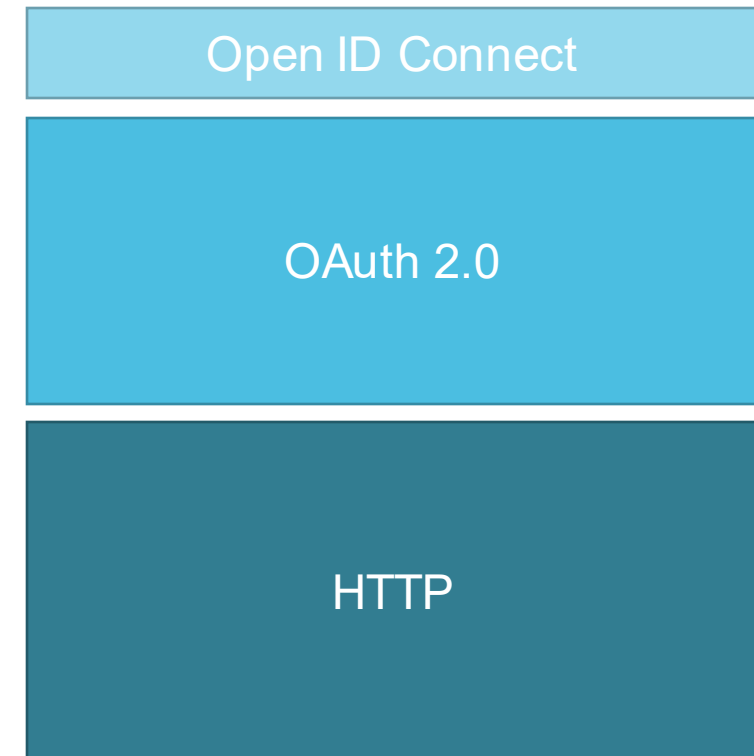
- Verschiedene Profile
  - Web-Browser SSO Profil ist OIDC am ähnlichsten
- Wesentlichster Unterschied: Assertions in XML
  - „Geschwätziger“ als JSON
  - Daher üblicherweise per POST versendet, schwieriger zu implementieren
- Weite Verbreitung bei Geschäftsanwendungen
  - Beispiel: ITMZ implementiert SAML 2.0-Schnittstelle für HDS-Account

```
<saml:Assertion ...
  ID="_d71a3a8e9fcc45c9e9d248ef7049393fc8f04e5f75" Version="2.0">
  <saml:Issuer>https://idp.example.org/SAML2</saml:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
  <saml:Subject>
    <saml:NameID ...>3f7b3dcf-1674-4ecd-92c8-1544f346baf8</saml:NameID>
    <saml:SubjectConfirmation ...>
      <saml:SubjectConfirmationData
        InResponseTo="aaf23196-1773-2113-474a-fe114412ab72"
        Recipient="https://sp.example.com/SAML2/SSO/POST"
        NotOnOrAfter="2004-12-05T09:27:05Z"/>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="..." NotOnOrAfter="...">
      <saml:AudienceRestriction>
        <saml:Audience>https://sp.example.com/SAML2</saml:Audience>
      </saml:AudienceRestriction>
    </saml:Conditions>
    ...
    <saml:AttributeStatement>
      <saml:Attribute ... Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1">
        <saml:AttributeValue xsi:type="xs:string">member</saml:AttributeValue>
        <saml:AttributeValue xsi:type="xs:string">staff</saml:AttributeValue>
      </saml:Attribute>
    </saml:AttributeStatement>
  </saml:Assertion>
```

# OPEN ID CONNECT 1.0

## Überblick

- Idee: Erweitere OAuth 2.0 für Authentifizierung
  - Standardisierter Endpunkt für Nutzerinformation
  - Token kann optional Nutzerinformationen beinhalten
    - ➔ kein separater Aufruf erforderlich
- Umsetzung: Spezifischer Scope „openid“
  - Token ist dann ID Token



# JSON WEB TOKEN (JWT)

## Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikd1b3JnIEhpbmtlbCI6Im1hdCI6MTYzOTMzOTAyMn0.De2irBdHg1alwh85Hqi_7qM5y1HUc9-0hXM0pVyN6So
```

## Decoded EDIT THE PAYLOAD AND SECRET

### HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

### PAYLOAD: DATA

```
{  "sub": "1234567890",  "name": "Georg Hinkel",  "iat": 1639339022}
```

### VERIFY SIGNATURE

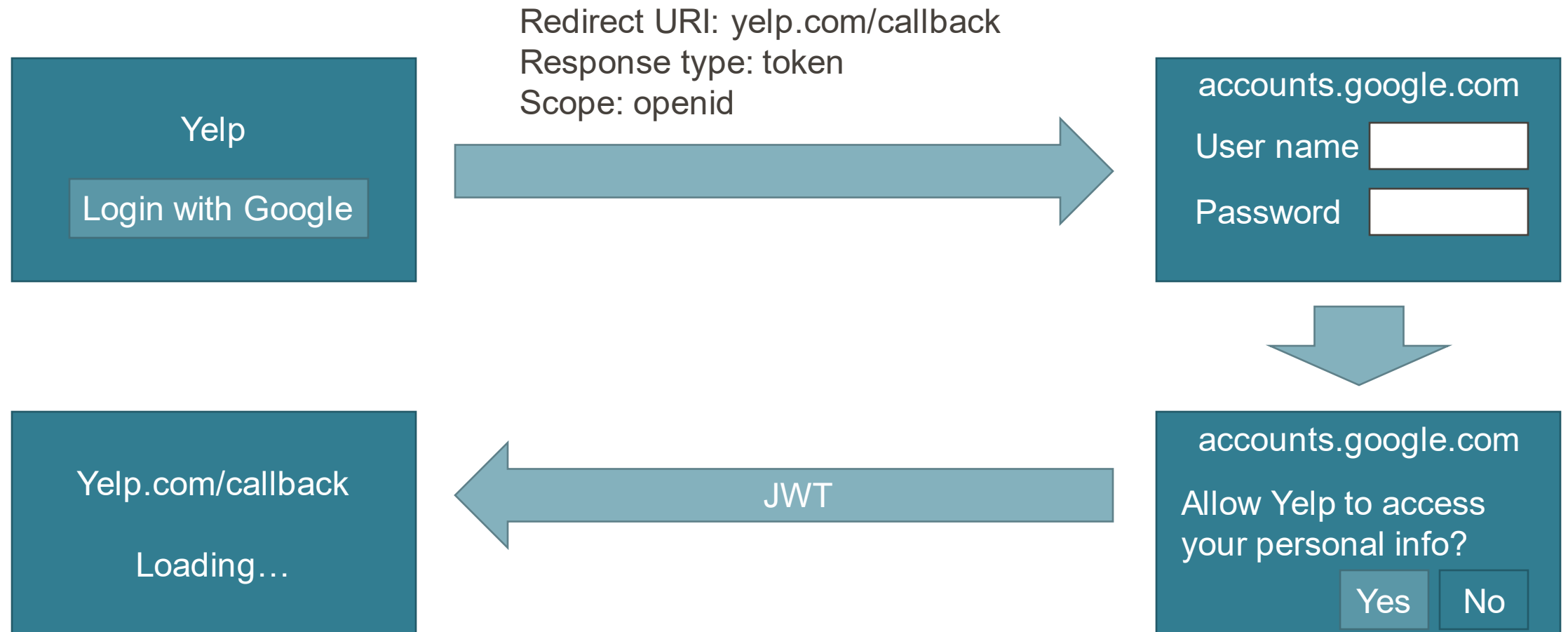
```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
    
) ☐ secret base64 encoded
```

Feldnamen kurz und  
standardisiert, um kurze  
Token erzeugen zu können

[<https://jwt.io>]

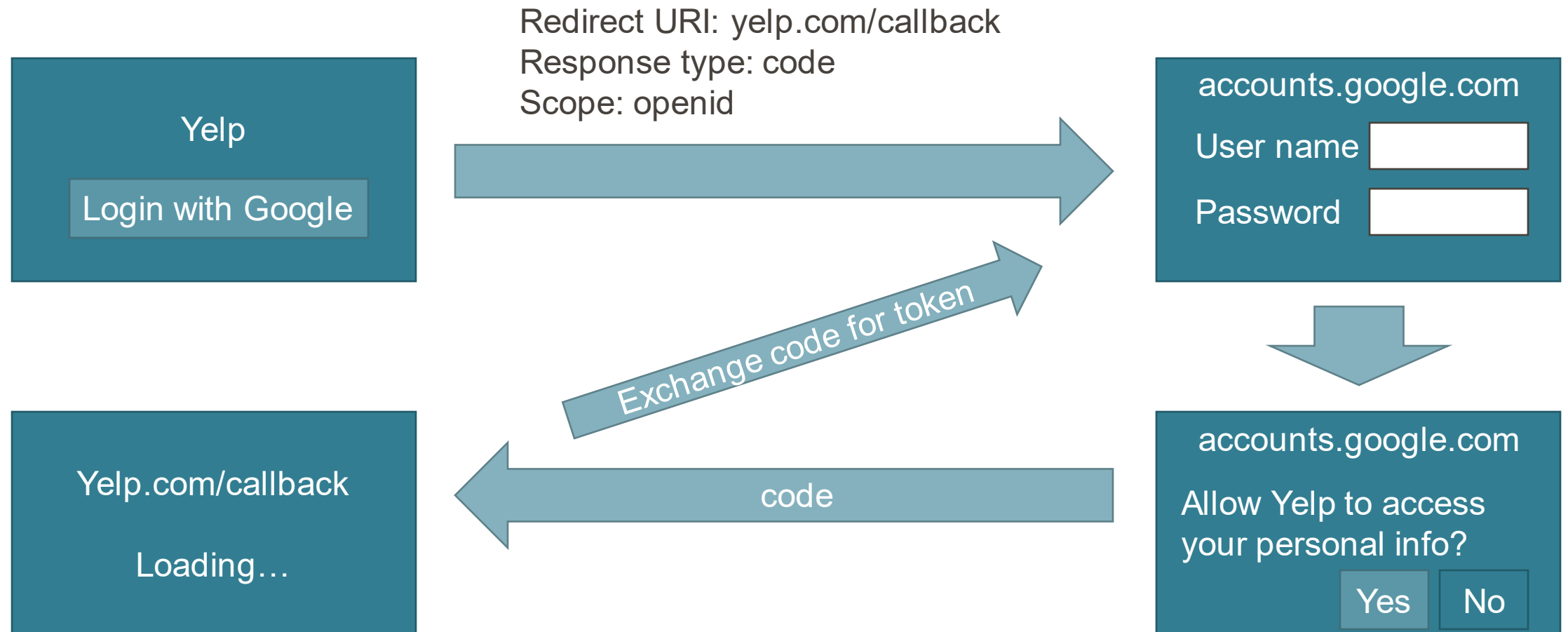
# OPEN ID CONNECT

## Implicit Flow



# OPEN ID CONNECT

## Authorization Code Flow



# PROOF KEY FOR CODE EXCHANGE (PKCE)

- Problem: Malware Im Browser könnte Code abfangen und gegen Access Token eintauschen wollen → insb. bei Single-Page-Anwendungen
- Lösung: PKCE
  1. Client würfelt Zufallszahl  $Ch_A$
  2. Sendet Hash der Zufallszahl  $H(Ch_A)$  zusammen mit ursprünglichem Request
  3. Gesamte Zufallszahl  $Ch_A$  wird zusammen mit Code mitgeschickt, um Access Token zu bekommen

- Derzeitige Empfehlung der IETF: Authentication Code Flow auch bei SPAs
  - Einschließlich PKCE
  - Aber Authentifizierung des Tokenaustauschs ohne Client Secret
  - Annahme: Angreifer, der Code abfangen kann, kann auch Client Secret abfangen

# AUTHORIZATION SERVER

## Beispiele

- Authorization Server großer Unternehmen
  - Google
  - Microsoft
  - Facebook, Twitter
  - GitHub, GitLab
  - ...
- Verzeichnisdienste von Cloud-Anbietern
- Separate Produkte
- Bibliotheken zur Erstellung eigener Authorization Server
- OpenID Foundation führt Listen
  - <https://openid.net/developers/certified/>



- Problem: Welchen Anbieter integrieren?
- Lösung: Viele, um häufigste Use Cases abzudecken, häufig mit expliziter UI
  - Login mit Facebook
  - Login mit Google
  - Login mit Twitter
  - ...
- Meist mit einem dedizierten Authorization Server, der mit anderen Authorization Servern federiert
  - Leitet Authorisierungsanfrage um
  - Fügt Autorisierungsinformationen hinzu
- Produkte
  - Keycloak, IdentityServer
  - Auth0, Okta

# IMPLEMENTIERUNG

Client (Beispiel: Angular, angular-oauth2-oidc)

- Modul hängt automatisch Token an jeden Backend-Request

```
OAuthModule.forRoot({  
  resourceServer: {  
    allowedUrls: ["https://..."],  
    sendAccessToken: true } }  
);
```

- Initialisierung

```
this.oauth.configure({  
  clientId: ...,  
  issuer: ...,  
  redirectUri: ...,  
  scope: ...,  
  responseType: "code"  
});  
this.oauth.loadDiscoveryDocumentAndLogin().then(() => {  
  this.oauth.setupAutomaticSilentRefresh();  
})
```

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.Authority = ...;
        options.Audience = ...;
        options.TokenValidationParameters ...
    });
```

Code zeigt direkte Auswertung des Tokens, weitere Möglichkeiten,  
um Authentifizierung bei Bedarf einzurichten

Token wird nur ausgewertet, wenn Endpunkt Authentifizierung erfordert ([Authorize])

- Konfiguration in Spring typischerweise gesteuert über `application.properties`
  - `spring.security.oauth2.resourceserver.jwt.issuer-uri`
  - `spring.security.oauth2.resourceserver.jwt.jwk-set-uri`
- Auswertung der Token wird in der Security Filter Chain eingetragen
  - `.oauth2ResourceServer().jwt();`
  - An dieser Stelle auch Auswahl der Requests, die authentifiziert werden müssen

- Delegierte Authentifikation / Autorisierung
  - Sie müssen kein Identity Provider sein, es sei denn Sie haben einen guten Grund dafür
- OAuth 2.0 / SAML 2.0 / OpenID Connect 1.0
  - Entscheidend ist meist, welche Infrastruktur bereits besteht
  - IETF-Empfehlung für neue Anwendungen: Open ID Connect
  - SAML 2.0 in Geschäftsanwendungen weit verbreitet



- Erläutern Sie den Begriff 2FA!
- Ein Unternehmen möchte, dass sich Kunden mit Ihrem Account bei <Social Media Plattform> anmelden können. Mit welchen Protokollen ließe sich das ermöglichen?
- Wofür wird das Protokoll OAuth 2.0 eingesetzt?
- Erläutern Sie den Unterschied zwischen Implicit Flow und Authorization Code Flow!
- Was ist ein JWT?
- Welche Flows gibt es in Open ID Connect 1.0 und was ist der Unterschied zwischen OAuth 2.0 und Open ID Connect 1.0?
- Ein Unternehmen setzt Open ID Connect 1.0 ein, um Anwendung in einer Systemlandschaft zu authentifizieren. Das Management hat beschlossen, in Zukunft nur noch 2-Faktor-Authentifizierung zuzulassen. Welche Server müssen dann angepasst werden? Inwieweit hängt die Antwort von den verwendeten Flows ab?