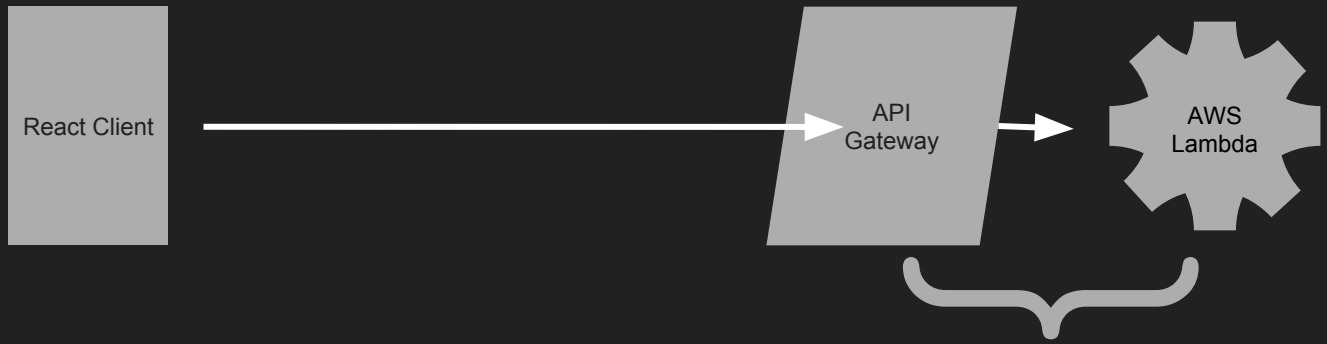
Access control in idiomatic React

Johan Peeters

Motivation

- Good software craftsmanship
- Good (enough) security



https://github.com/softwarewolves/riders.git
https://github.com/JohanPeeters/riders.git

https://github.com/JohanPeeters/rides-api

http://localhost:3000
https://ride-sharing.ml

https://3o7a5pnqt7.execute-api.eu-west-1.amazonaws.com/prod/rides

About Johan

- Security architect
- Founder of secappdev.org
- Consultancy and training
- Bespoke development
- Lecturer at EhB

https://www.johanpeeters.com
@YoPeeters
yo@johanpeeters.com

API keys

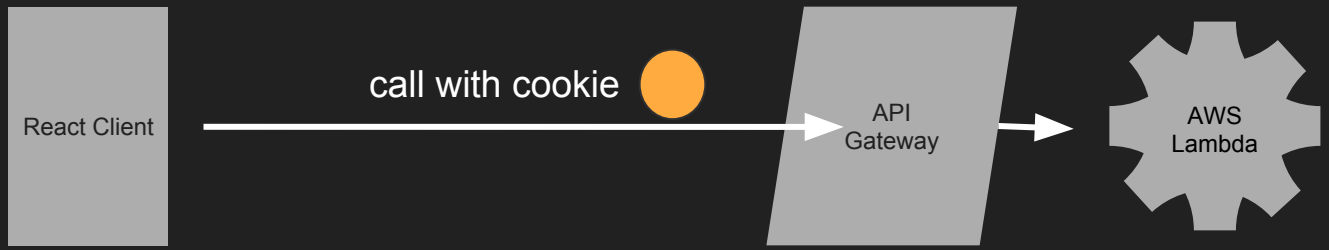
- issued to the app developer
- great to stop Exhaustion of Funds (EoF) attacks
  - throttle limits
  - quota
- great for analytics
- OK for pay-per-use APIs if stakes are low
- pretty useless for access control
  - key shared across many instances of the client
  - key is available on a public client
  - revocation is problematic

CORS

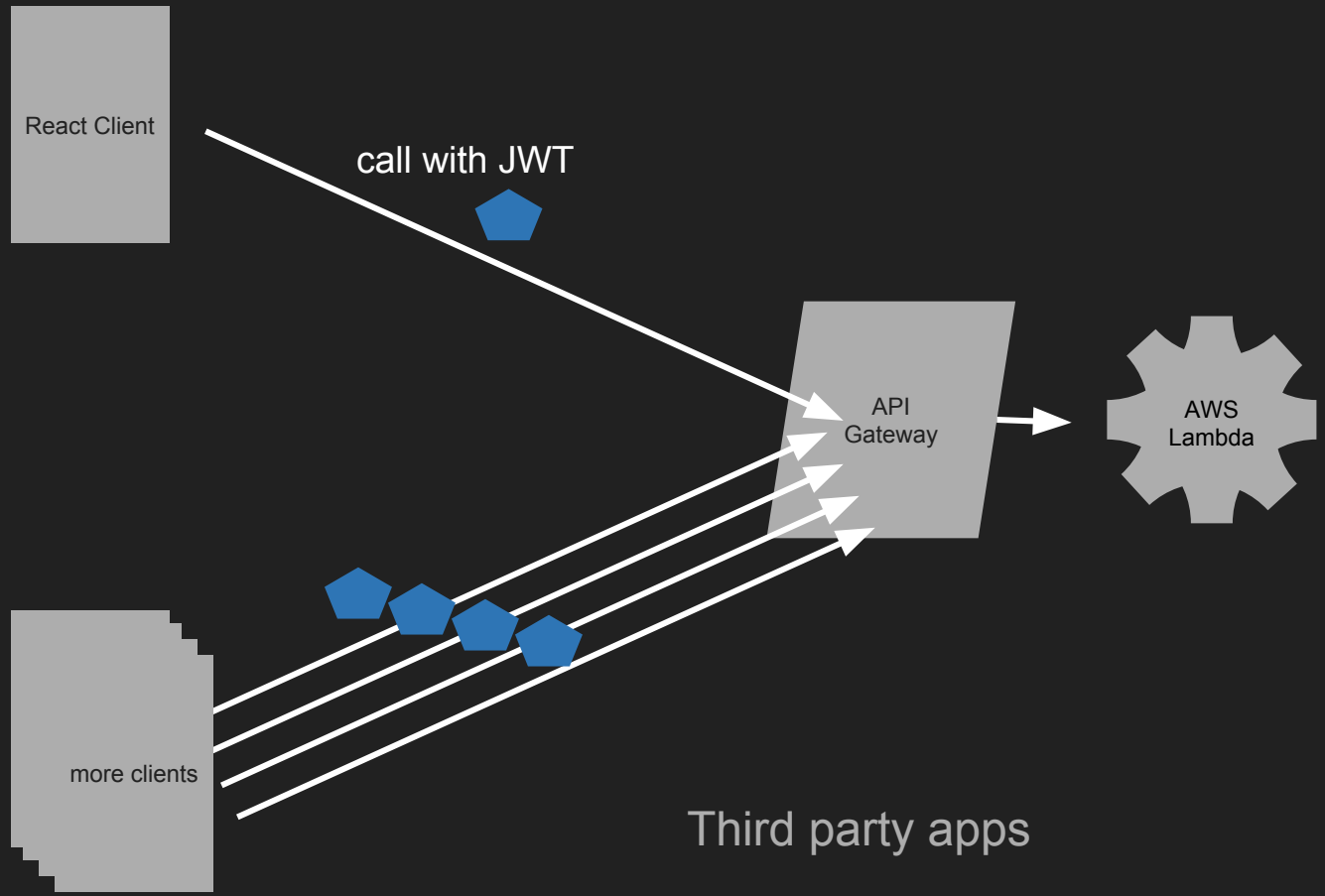
- relaxes the Same Origin Policy to allow cross-origin calls
- `Access-Control-Allow-*` response headers
- frequent source of developer bewilderment
  - using the same origin for client and API (i.e. a first party app) solves this
  - but, if you can do this, most of this workshop is irrelevant - see below
- access control based on origin of client
  - origin can easily be faked outside the browser
  - protects the client, not the API
- CORS leaves the API largely unprotected
  - white-listing origin, methods and headers affords some small measure of protection
  - just bouncing back `Access-Control-Allow-Origin *` wastes that opportunity
  - reflecting the origin turns out to be worse than useless (https://ejj.io/misconfigured-cors)

Why not use cookies?

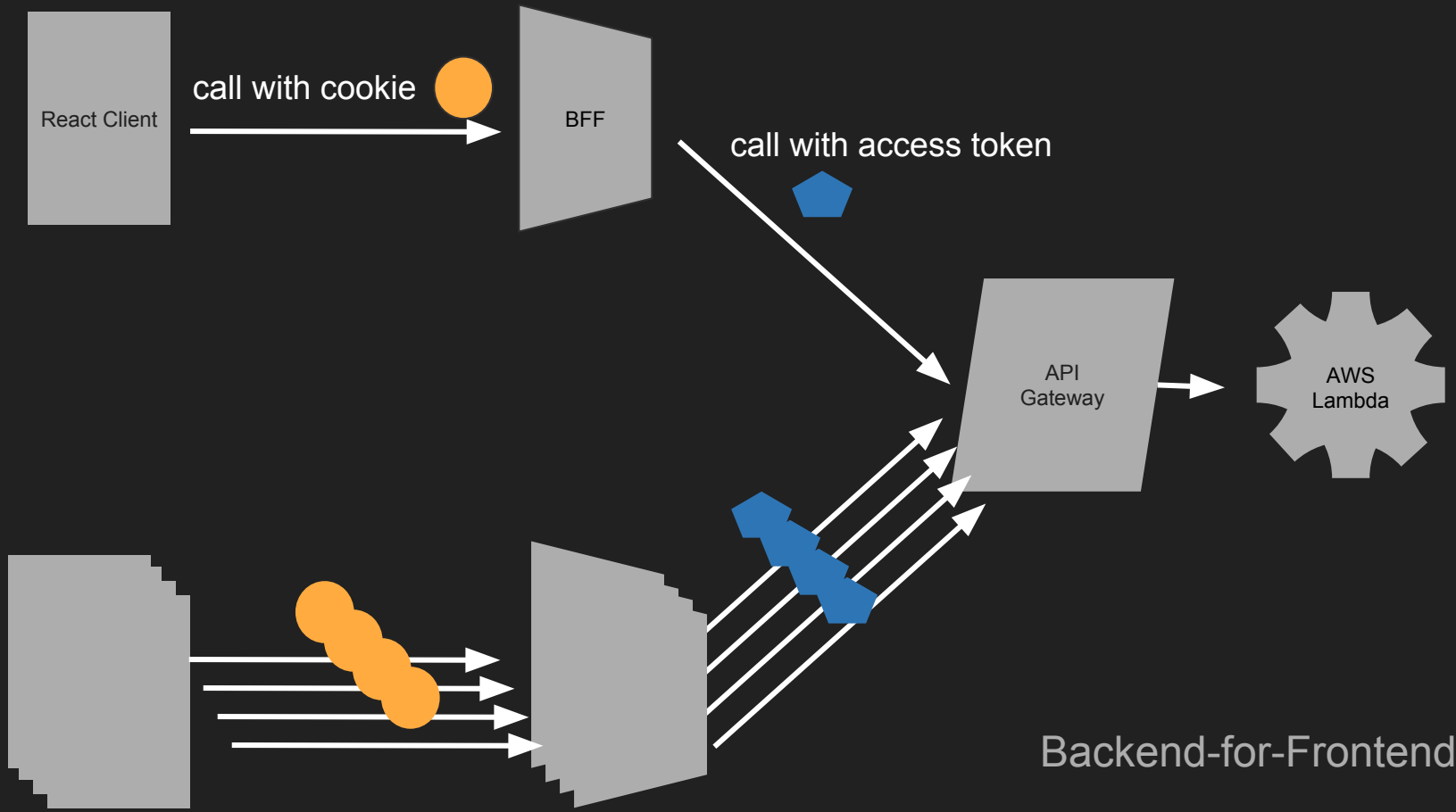
- recommended for first-party apps
  - draft IETF BCP ‘OAuth 2.0 for Browser-based Apps’
  - https://datatracker.ietf.org/doc/draft-ietf-oauth-browser-based-apps/
  - fewer moving parts, smaller attack surface
  - caveat: setting the cookie is not trivial
- proposal for BFF to interact with authorization server
  - https://t.co/71pc4EFHDd
  - the reverse proxy handles the OAuth/OIDC flows
    - confidential client
    - tokens are harder to steal because on the back-end
  - however, more moving parts, more complex to deploy



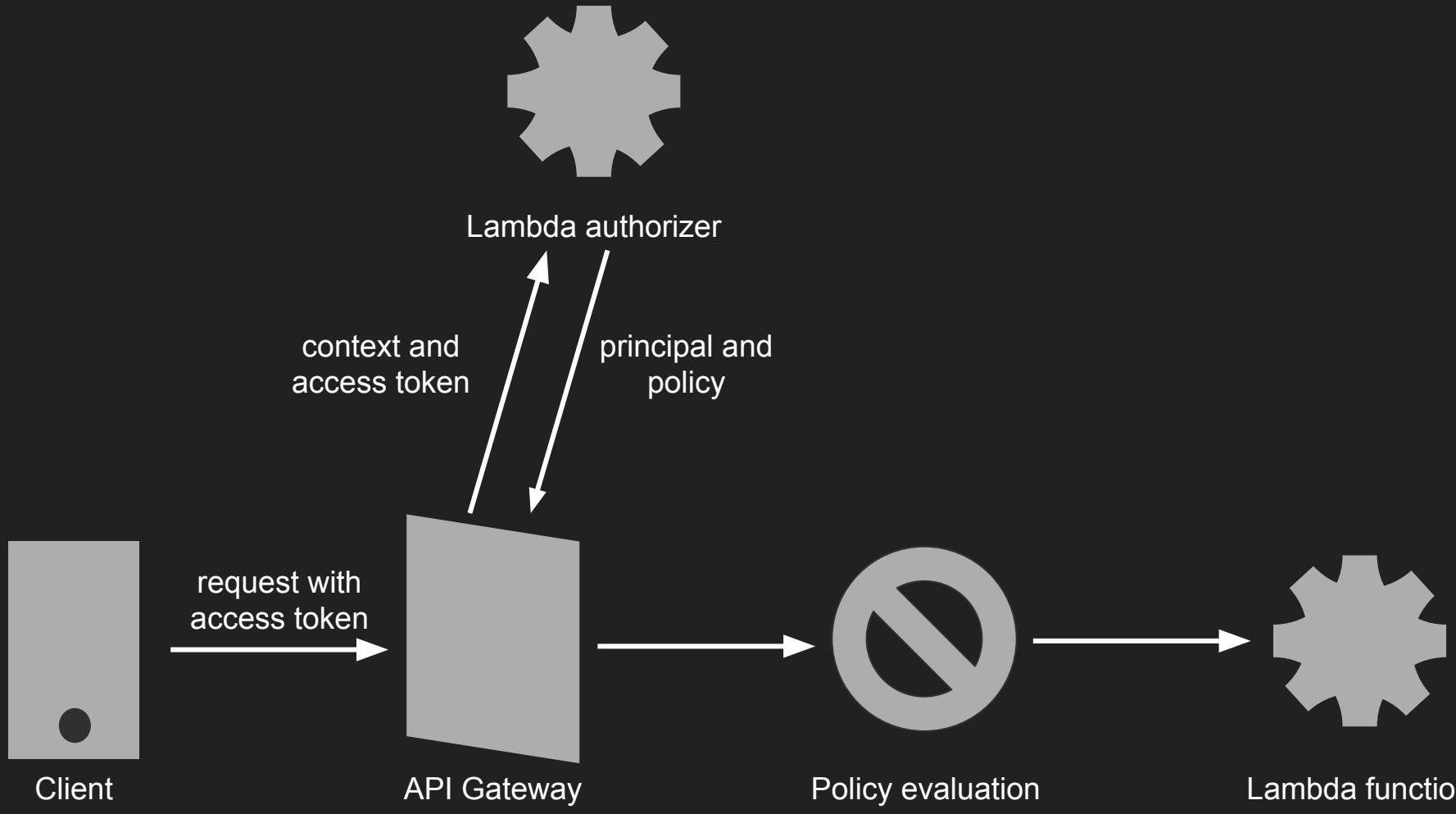
First party app

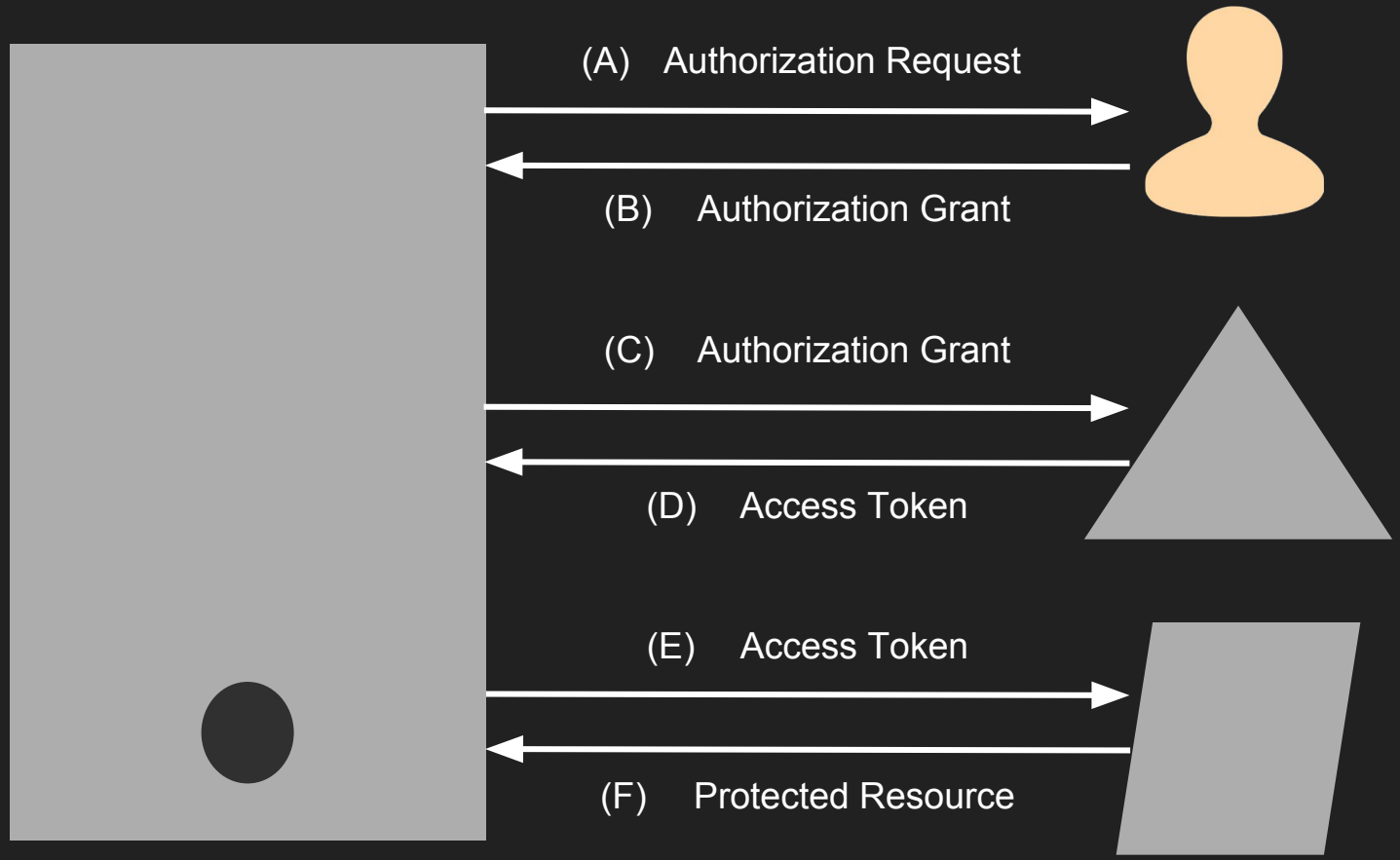


Third party apps



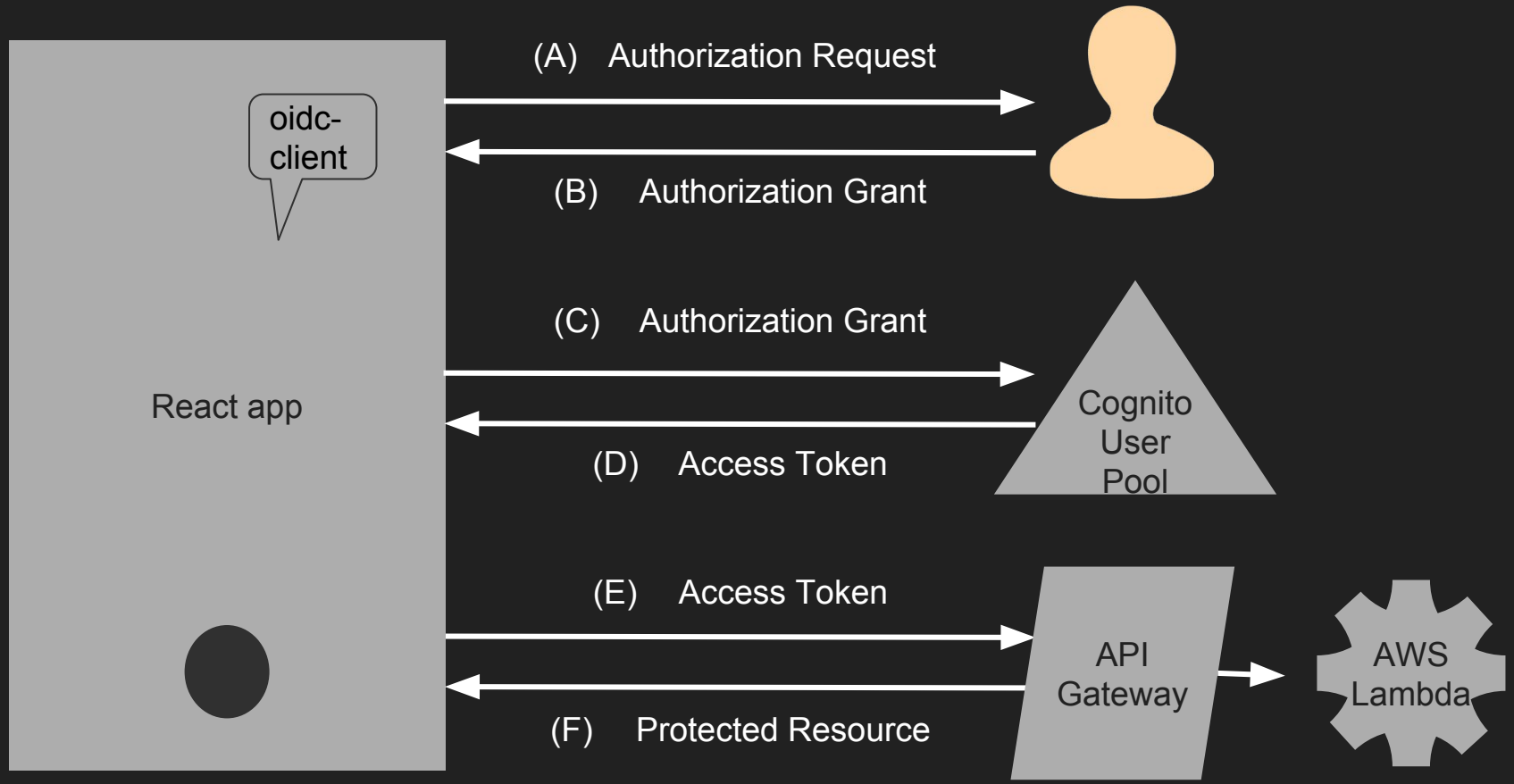
Backend-for-Frontend (BFF)


Lambda authorizer
context and
access token
principal and
policy
request with
access token
Client
API Gateway
Policy evaluation
Lambda function



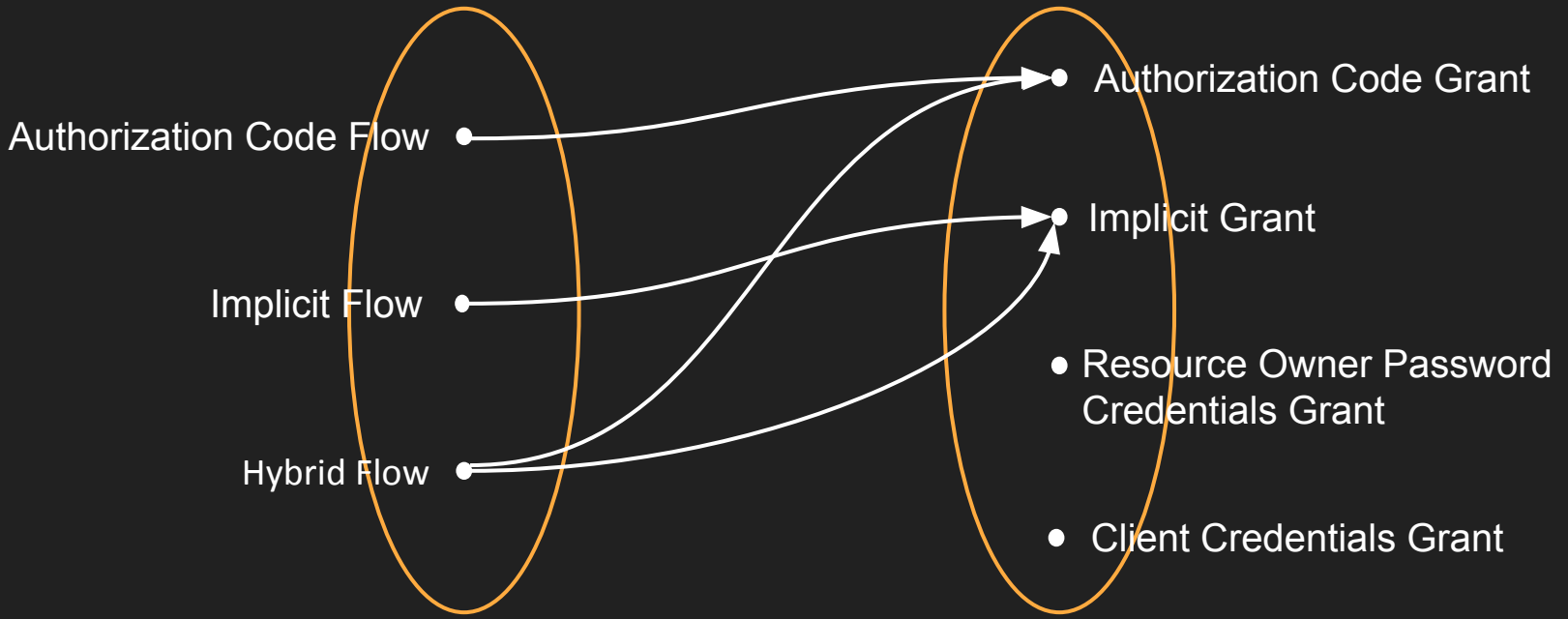
Abstract OAuth Protocol Flow

From RFC 6749 ©IETF



Concrete components

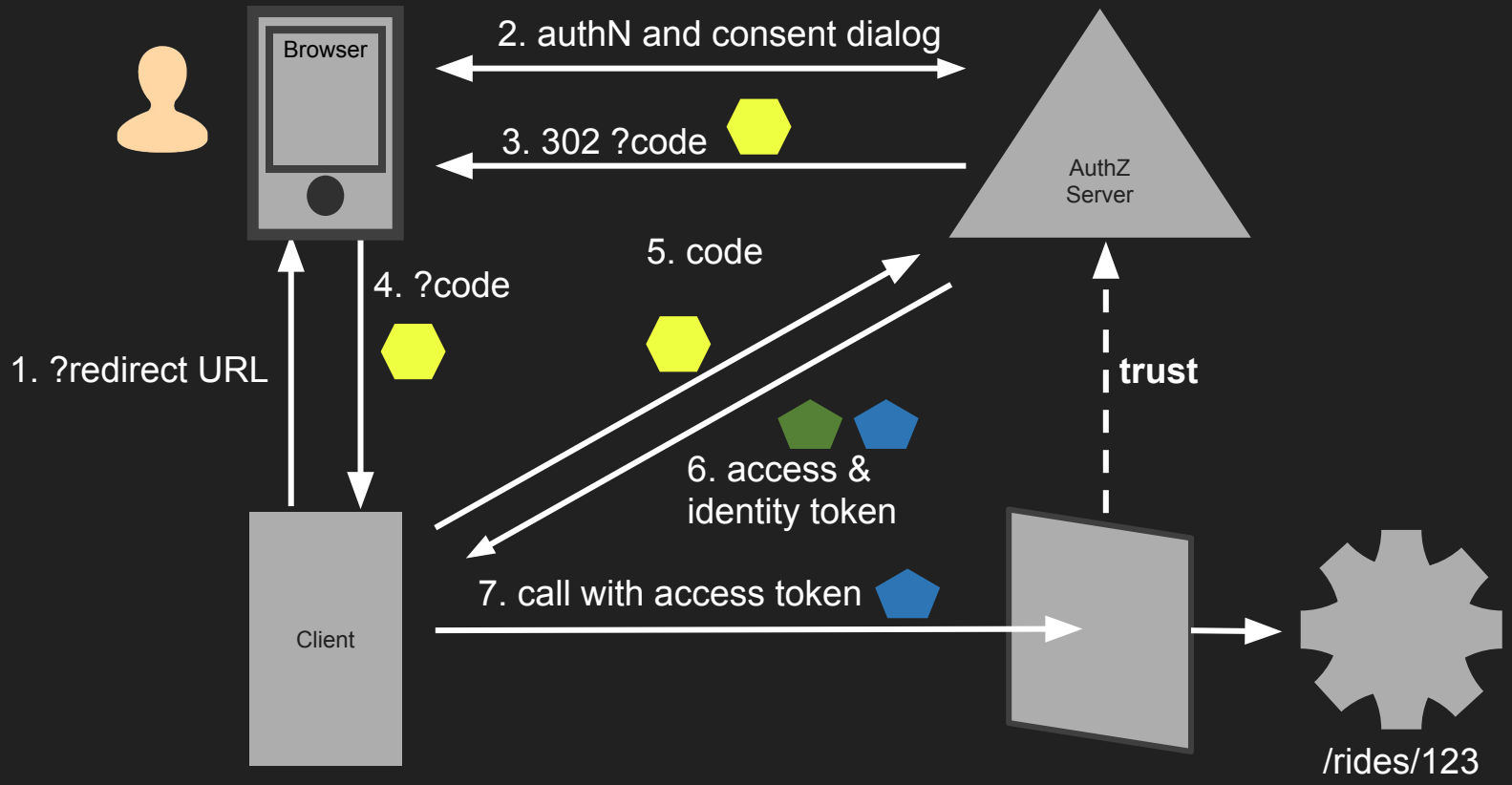
Historic OAuth authorization grants/OIDC flows



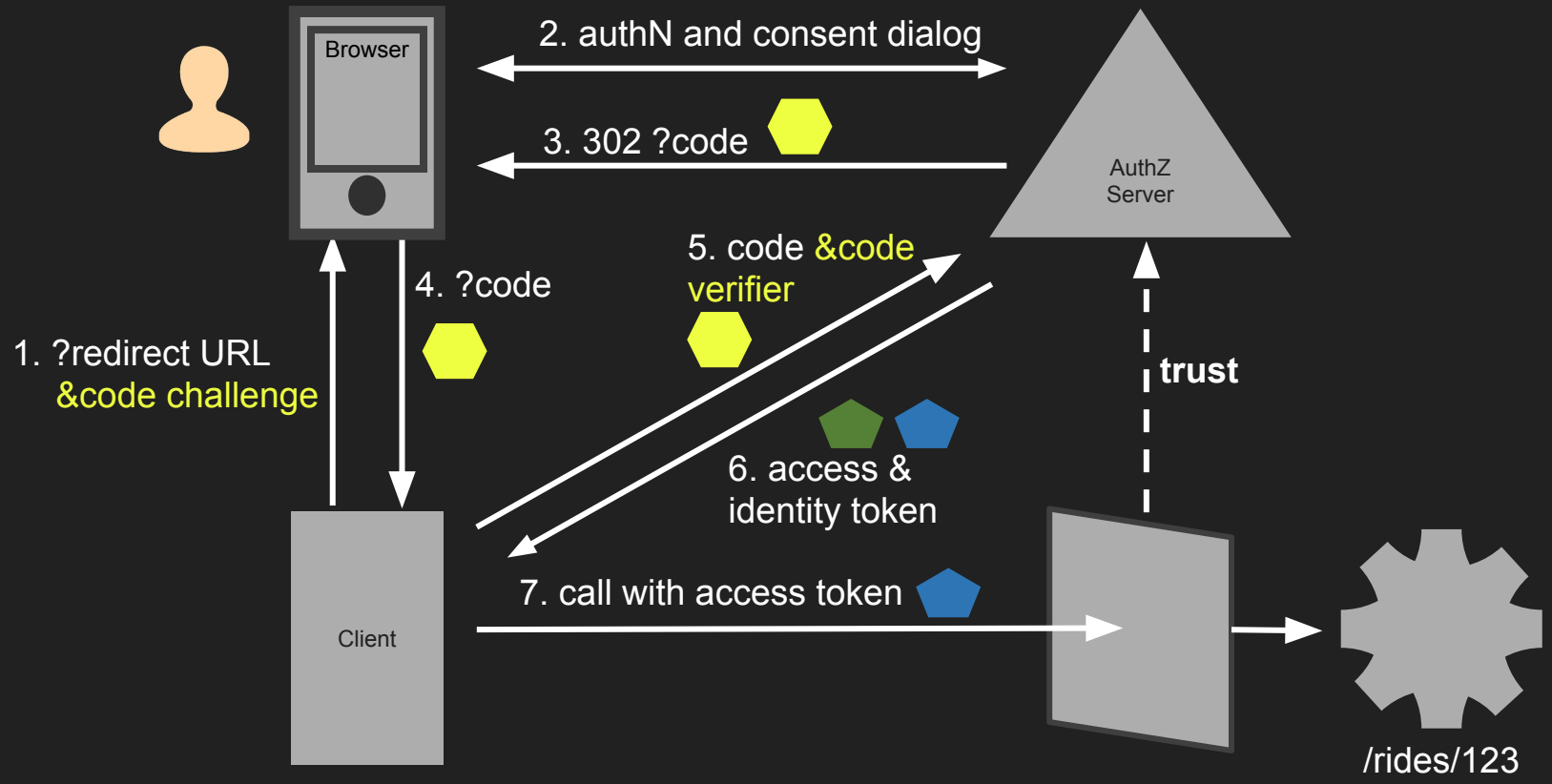
OIDC flows

OAuth 2.0 grants

Authorization Code Flow



Authorization Code Flow with PKCE



Challenge #1: authenticate with the AS

Acceptance criteria:

- app opens login page on AS
- AS sends code back in a query parameter to the redirect URI

What redirect URIs does the AS accept?

- `http://localhost:3000`,
- `http://localhost:3000/callback`,
- `https://ride-sharing.ml/callback`,
- `https://ride-sharing.tk`

Challenge #2: exchange code for token

Acceptance criteria:

- the authorization server sends back security tokens
- the code is not in the browser history

Hints:

- UserManager has a method for the job
- one of the unused React components may come in handy
- leverage React router

Challenge #3: place app into an authenticated state

Acceptance criteria:

- after logging in, a logout button shows
- menu items are enabled

Challenge #4: logging out

Acceptance criteria:

- logout button is replaced by a login button
- menu items are disabled as appropriate
- the user must re-authenticate when logging back in
- no tokens in local or session storage

Hints:

- look at the Cognito logout endpoint docs
- use `http://localhost:3000` as logout URI

Challenge #5: remain authN’ed across page reloads

Acceptance criteria:

when you log in and reload the page, you are still logged in

Challenge #6: calls to protected APIs succeed

Acceptance criteria:

- you can add a ride
- you can delete your own rides
- you can edit and update your own rides

Hint:

the various methods have their specific scopes:

- rides/create
- rides/delete
- rides/update

Evaluation

What have you learned?

What did you expect to learn and didn’t?

What is still unclear?

References

- OAuth 2.0 for native apps: https://datatracker.ietf.org/doc/rfc8252/
- OAuth 2.0 for browser-based apps best current practice: https://datatracker.ietf.org/doc/draft-ietf-oauth-browser-based-apps/
- OAuth 2.0 security best current practice: https://datatracker.ietf.org/doc/draft-ietf-oauth-security-topics/
- OAuth 2.0 Threat Model and Security Considerations: https://tools.ietf.org/html/rfc6819