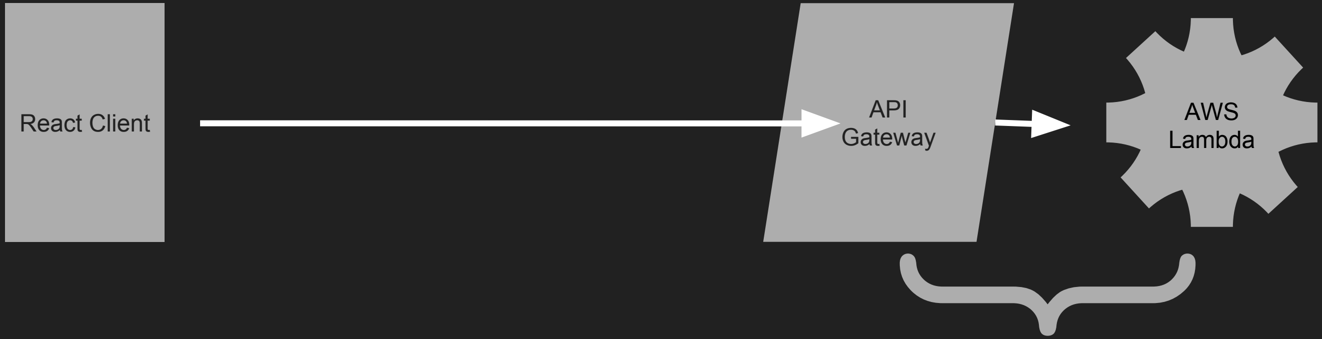
API access control

Michael Boeynaems - Johan Peeters



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 Michael

- Independent cyber security consultant
- Lecturer at AP
- Security architect @ Colruyt Group

https://www.portasecura.com

https://www.linkedin.com/in/michaelboeynaems

michael.boeynaems@portasecura.com

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

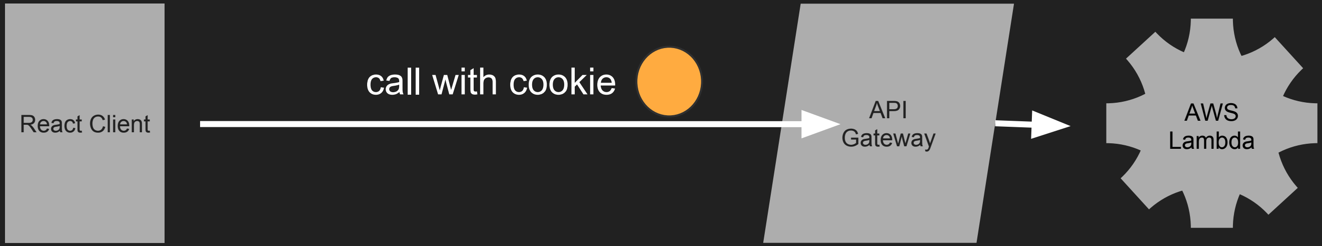
- issued to the app developer
- great to stop Exhaustion of Funds (EoF) attacks
  - throttle limits
  - quota
- great for analytics
- OK for pay-per-use APIs if stakes are low
- pretty useless for access control
  - key shared across many instances of the client
  - key is available on a public client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

CORS

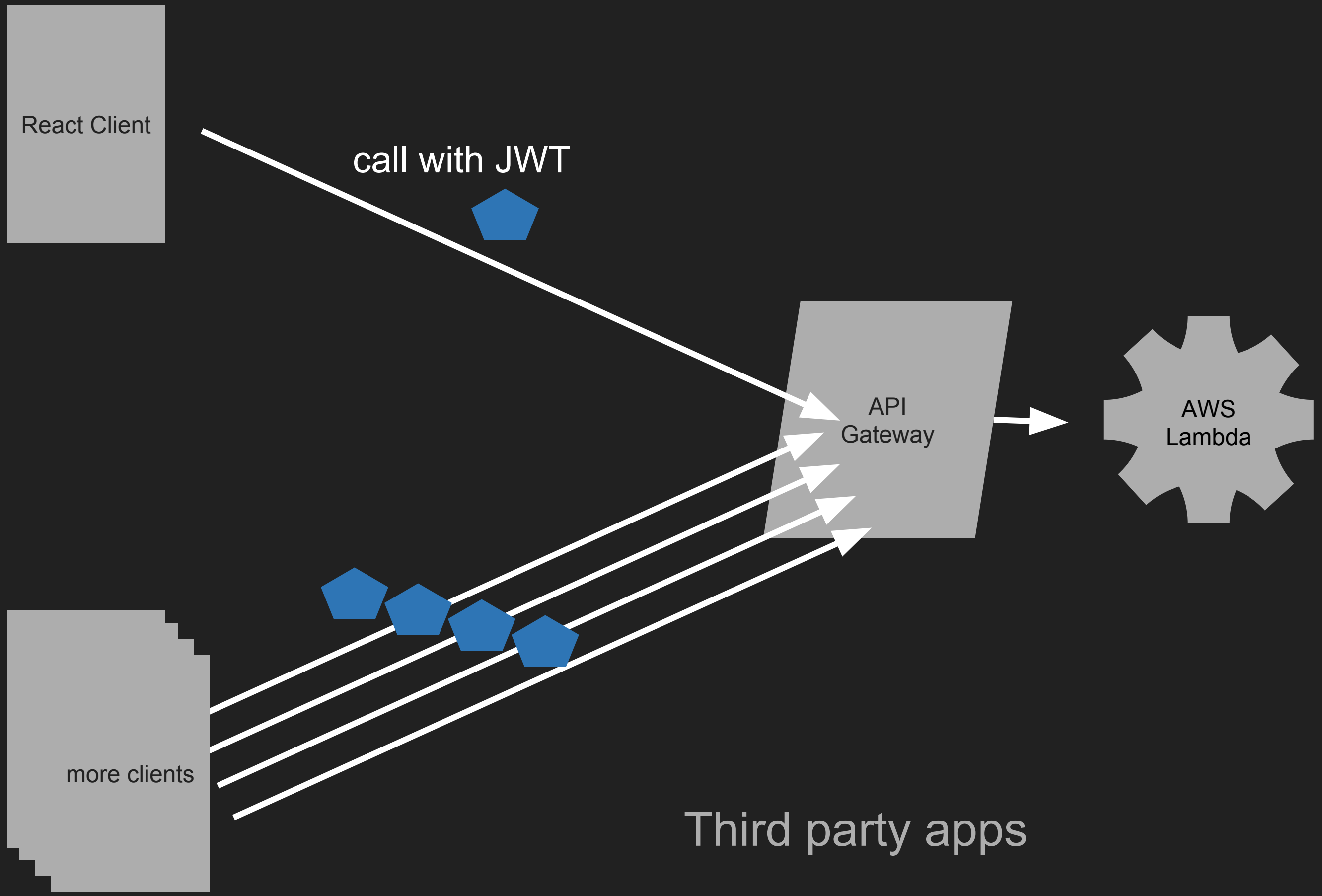
- relaxes the Same Origin Policy to allow cross-origin calls
- `Access-Control-Allow-*` response headers
- frequent source of developer bewilderment
  - using the same origin for client and API (i.e. a first party app) solves this
  - but, if you can do this, most of this talk is irrelevant - see below
- 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?

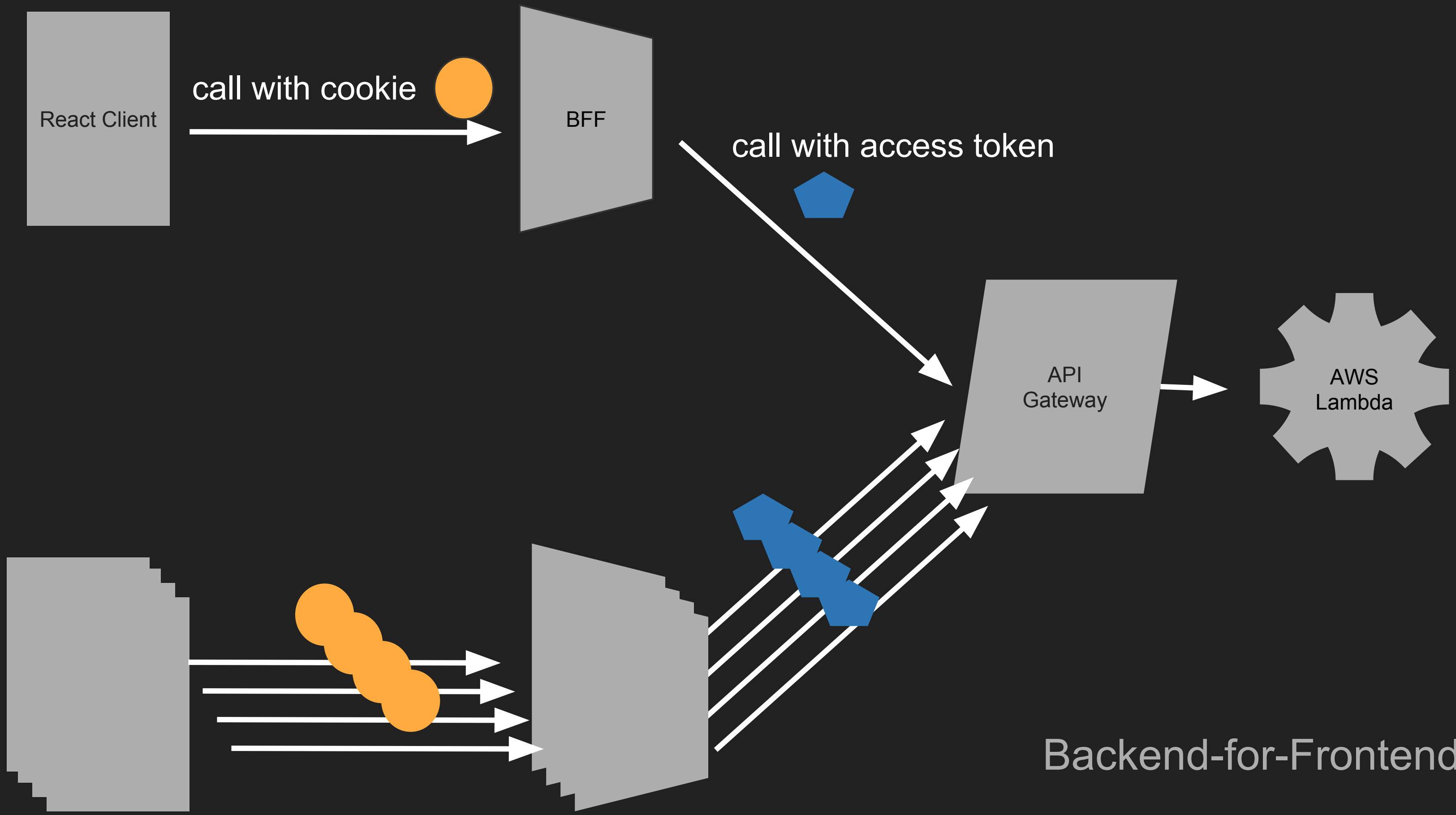
- recommended for first-party apps
  - draft IETF BCP ‘OAuth 2.0 for Browser-based Apps’
  - https://datatracker.ietf.org/doc/draft-ietf-oauth-browser-based-apps/
  - fewer moving parts, smaller attack surface
  - caveat: setting the cookie is not trivial
- proposal for BFF to interact with authorization 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



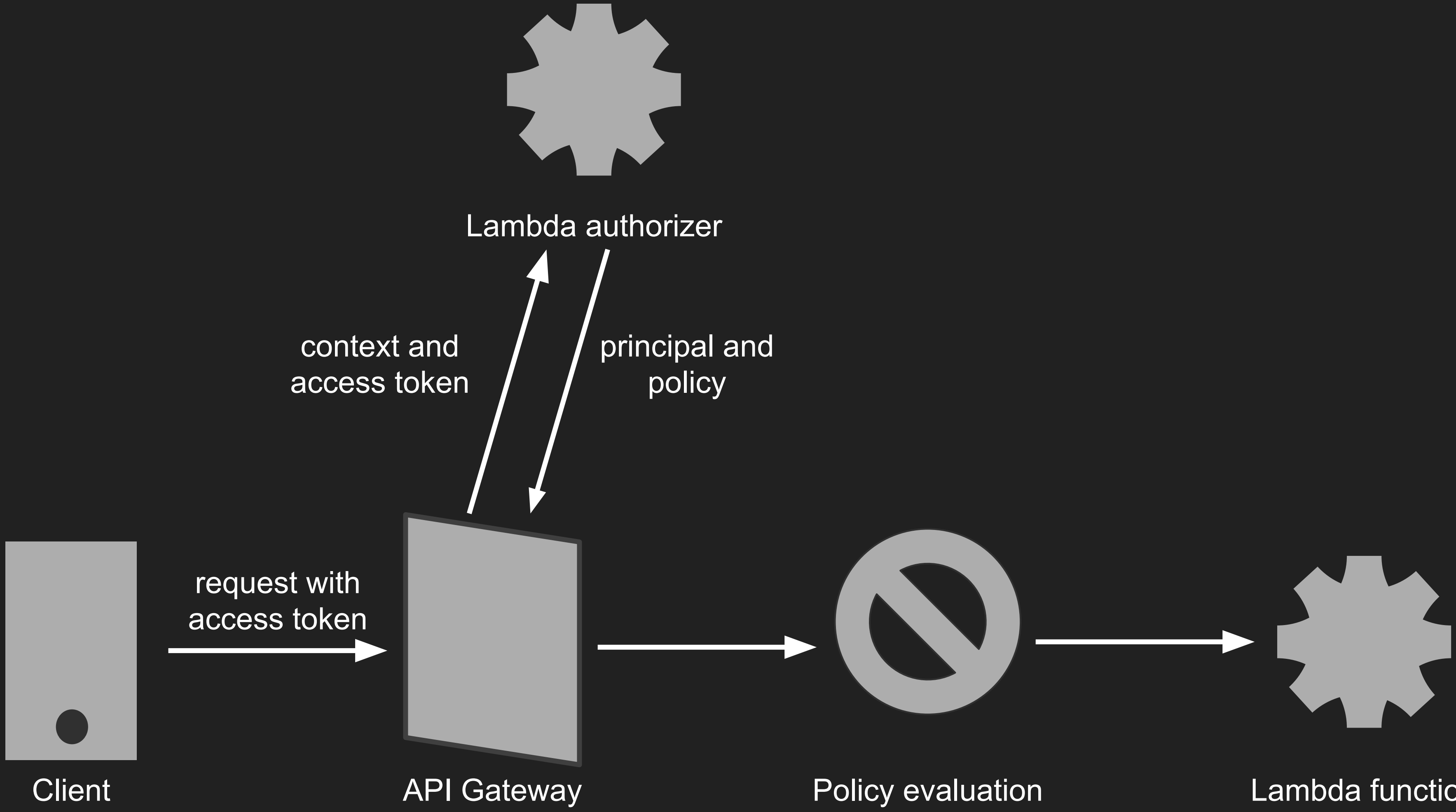
First party app

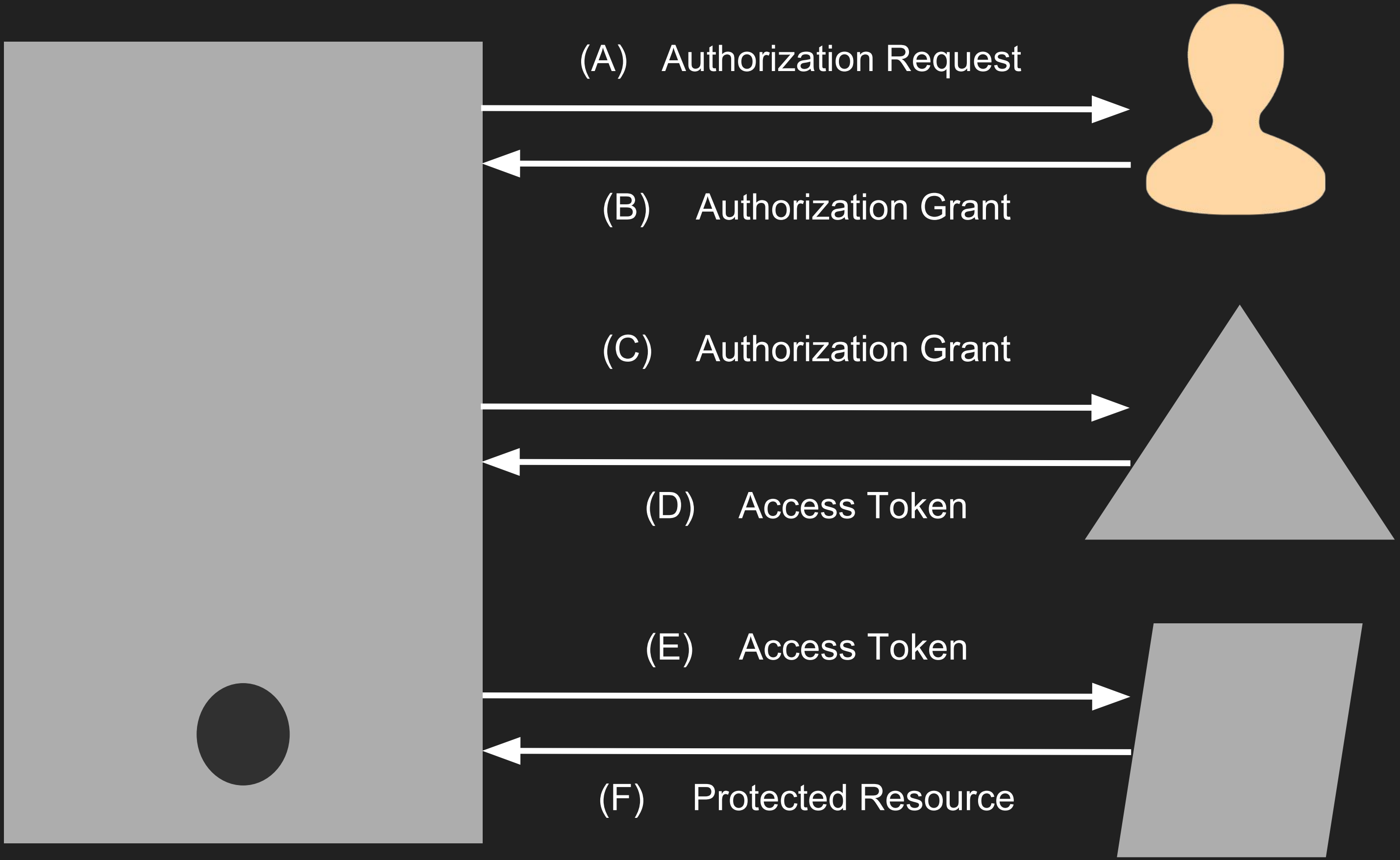


Third party apps

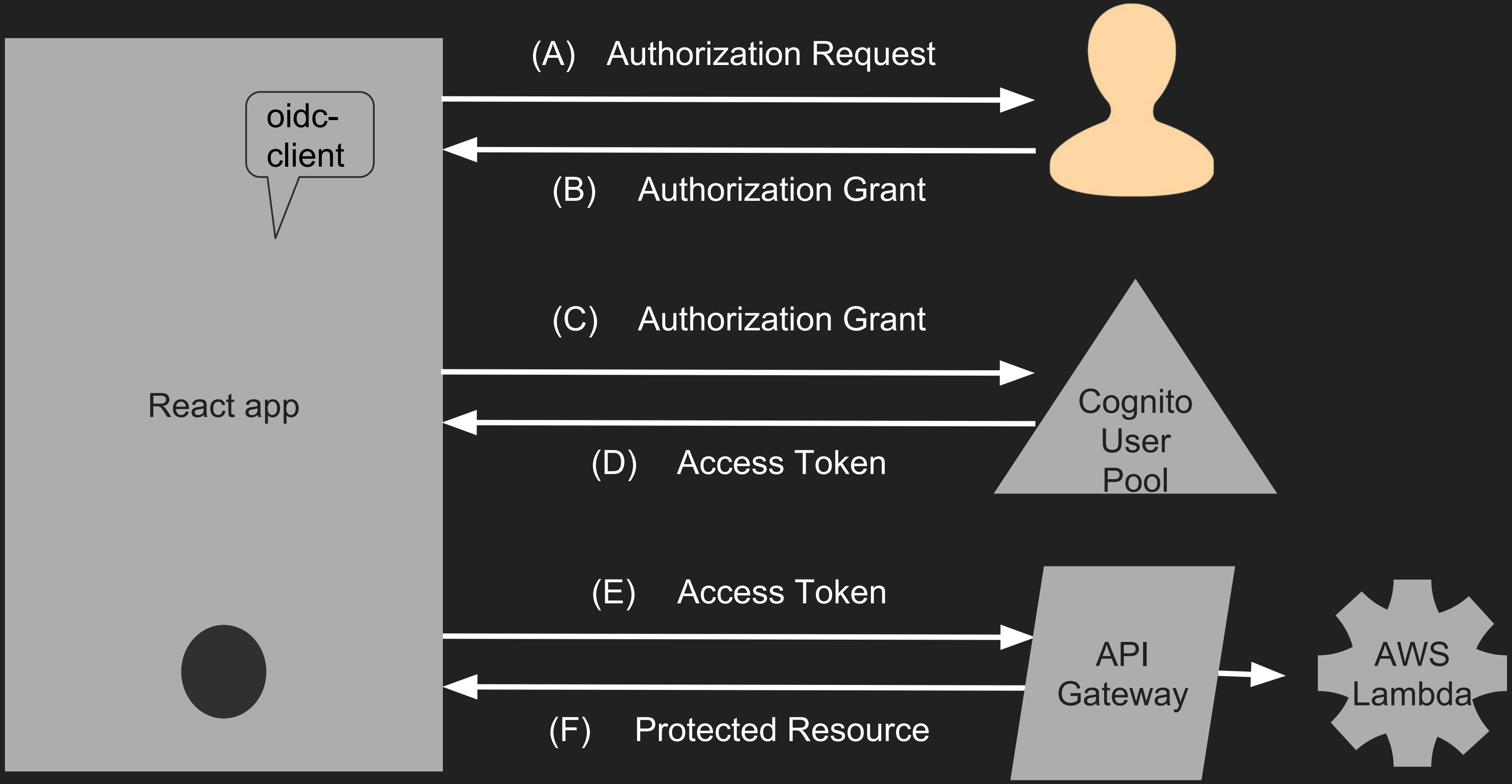


Backend-for-Frontend (BFF)


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

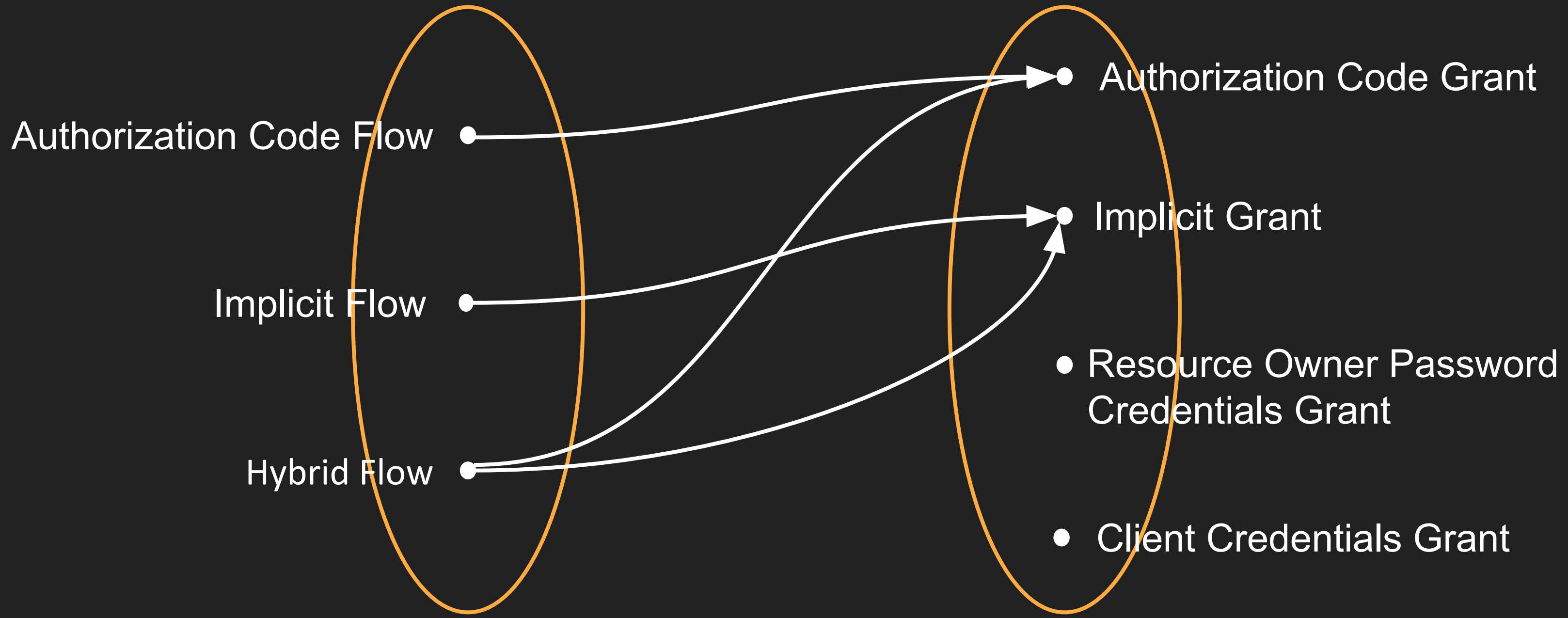


Abstract OAuth Protocol Flow



Concrete components

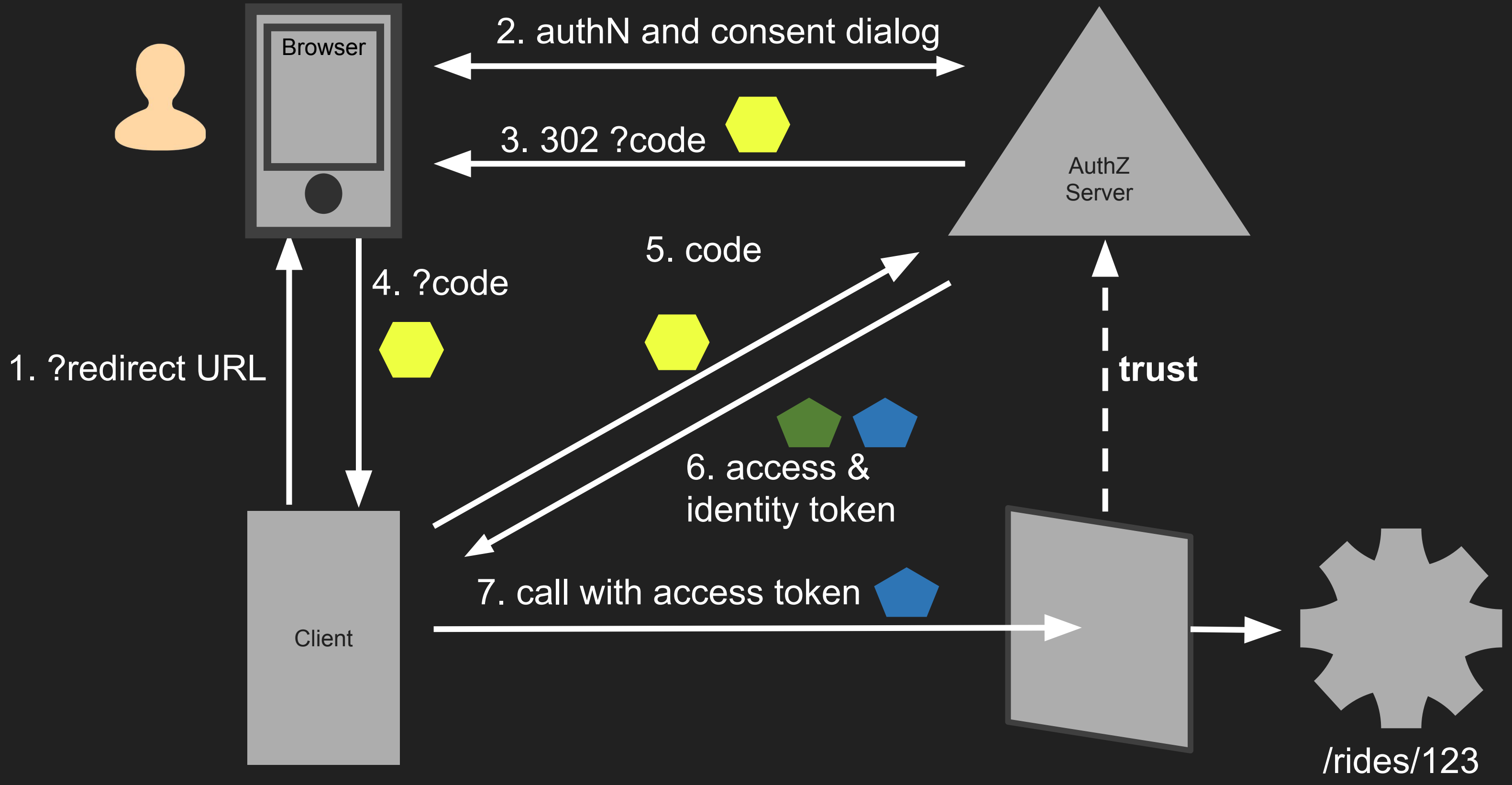
Historic OAuth authorization grants/OIDC flows



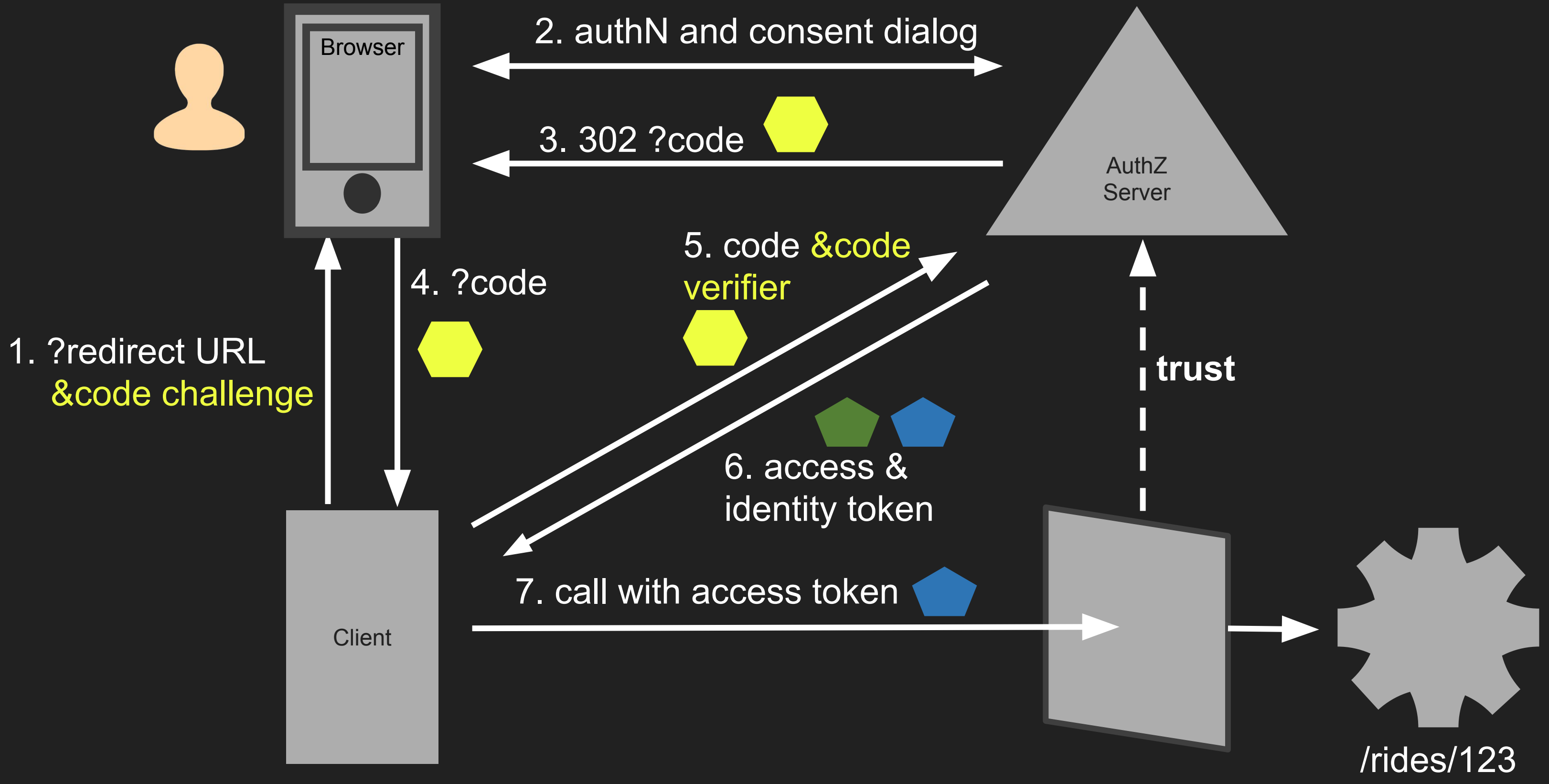
OIDC flows

OAuth 2.0 grants

Authorization Code Flow



Authorization Code Flow with PKCE


Browser
2. authN and consent dialog
3. 302 ?code
AuthZ Server
4. ?code
5. code &code verifier
1. ?redirect URL &code challenge
trust
6. access & identity token
Client
7. call with access token
/rides/123

When to use which flow?

Client type	**Flow**	**Refresh token allowed?**
Unattended authentication	Client Credentials	No
Single Page Application	Authorization Code with PKCE	No
Backend web application	Authorization Code with PKCE	Yes
Native application	Authorization Code with PKCE via external user-agent	Yes

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/