ThemiStruct
テミストラクト

認証標準技術の必要性と最新動向

株式会社オージス総研
事業開発本部 テミストラクトソリューション部
篠原　奨

なぜ標準技術を使うのか？

標準技術を使うメリット

ロ セキュリティ向上

- 標準技術は脆弱性に関する考慮がなされているため安全に利用できる
  - 独自実装を行う場合、実装に脆弱性がないことを担保するのは難易度が高く、コストもかかる
- 標準技術は危殆化したら対策が取られるため、継続的に安全性を確保できる
  - 脆弱性が発覚した場合には追加仕様や次バージョンが策定される等の対策が取られることが多いため、常に安全な状態で利用できる

標準技術を使うメリット

□ 相互接続性の向上

- インターフェース仕様が固まっているためアプリケーション間の接続を容易に行うことができる
  - アプリケーションが標準技術通りに実装されていればアプリケーション間でのすり合わせが最小限で済む
  - サービスの利用、パッケージ製品の導入・連携も容易に行える
- アプリケーション・認証基盤のリプレースが容易に行える
  - 独自実装アプリケーションとの連携が存在する場合、リプレース時に独自実装を引き継がなければいけない負の連鎖に陥りやすいので注意が必要

標準技術を使うメリット

ロ 開発工数の削減

- インターフェース・動作仕様が固まっているので設計・検討時間を短縮できる
- 既存ライブラリが存在する場合、利用することで開発工数・バグ混入率の低減を図ることができる

だが、漠然と標準技術を利用していればメリットを享受できるわけではない

標準技術選定時の考慮点

ロ 用途に沿った技術を選択することが重要である

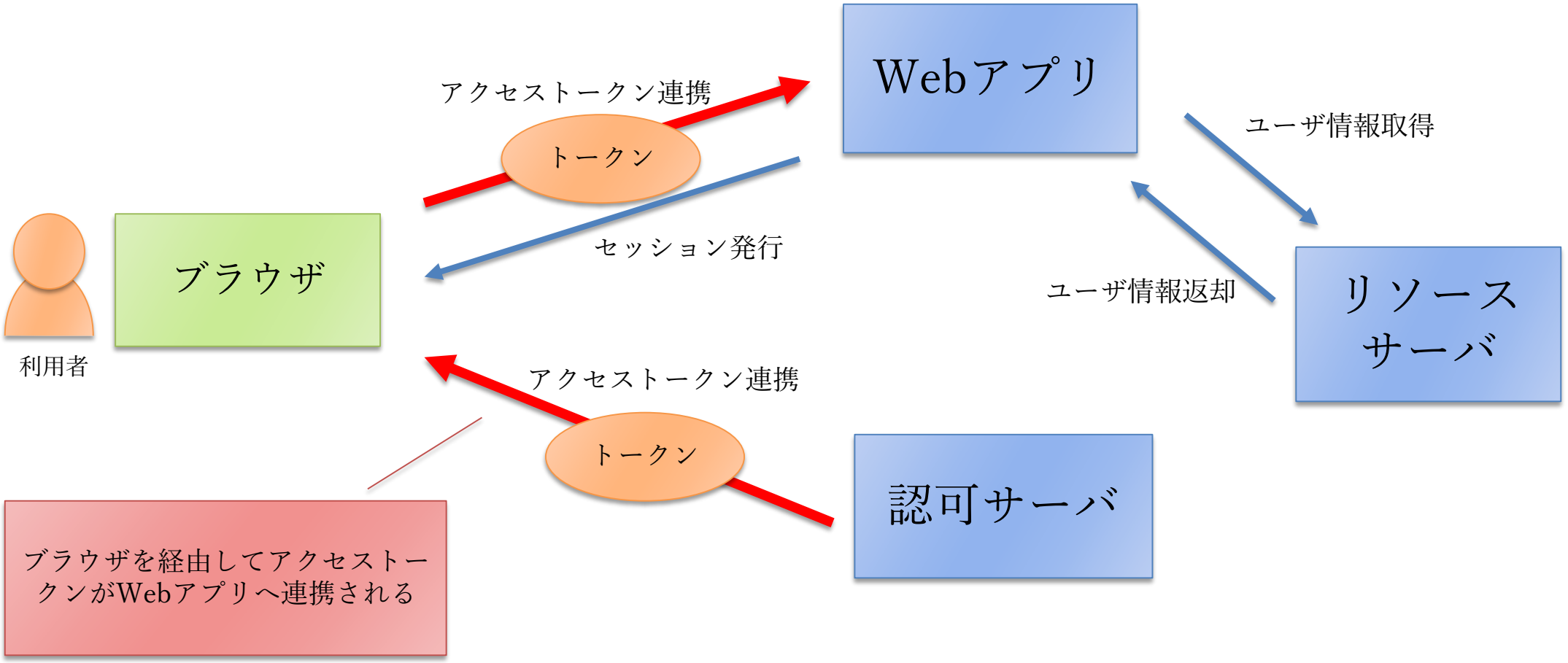
- 誤った用途で標準技術を利用すると、おもわぬ脆弱性を生む可能性がある
- 例：OAuth認証
  - API認可に利用すべき「OAuth」をアプリケーション間の認証連携（シングルサインオン）に利用したことにより脆弱性が発生するケースがある

OAuthとは

- **アクセストークンを元にAPIの認可を行う仕様**
  - 一般的にリソースアクセス時に認証は行われず、アクセストークンを保持していることでリソースにアクセスできる
- **アクセストークンには以下のような情報が紐づいている（一部抜粋）**
  - scope（権限範囲）
  - sub（ユーザ識別子）
  - exp（有効期限）
  - aud（アクセストークン利用対象リソース）
  - iss（発行元）
  - iat（発行日時）
- **リソースサーバはこれらの情報をもとにリソースの返却可否・返却対象を判断する**

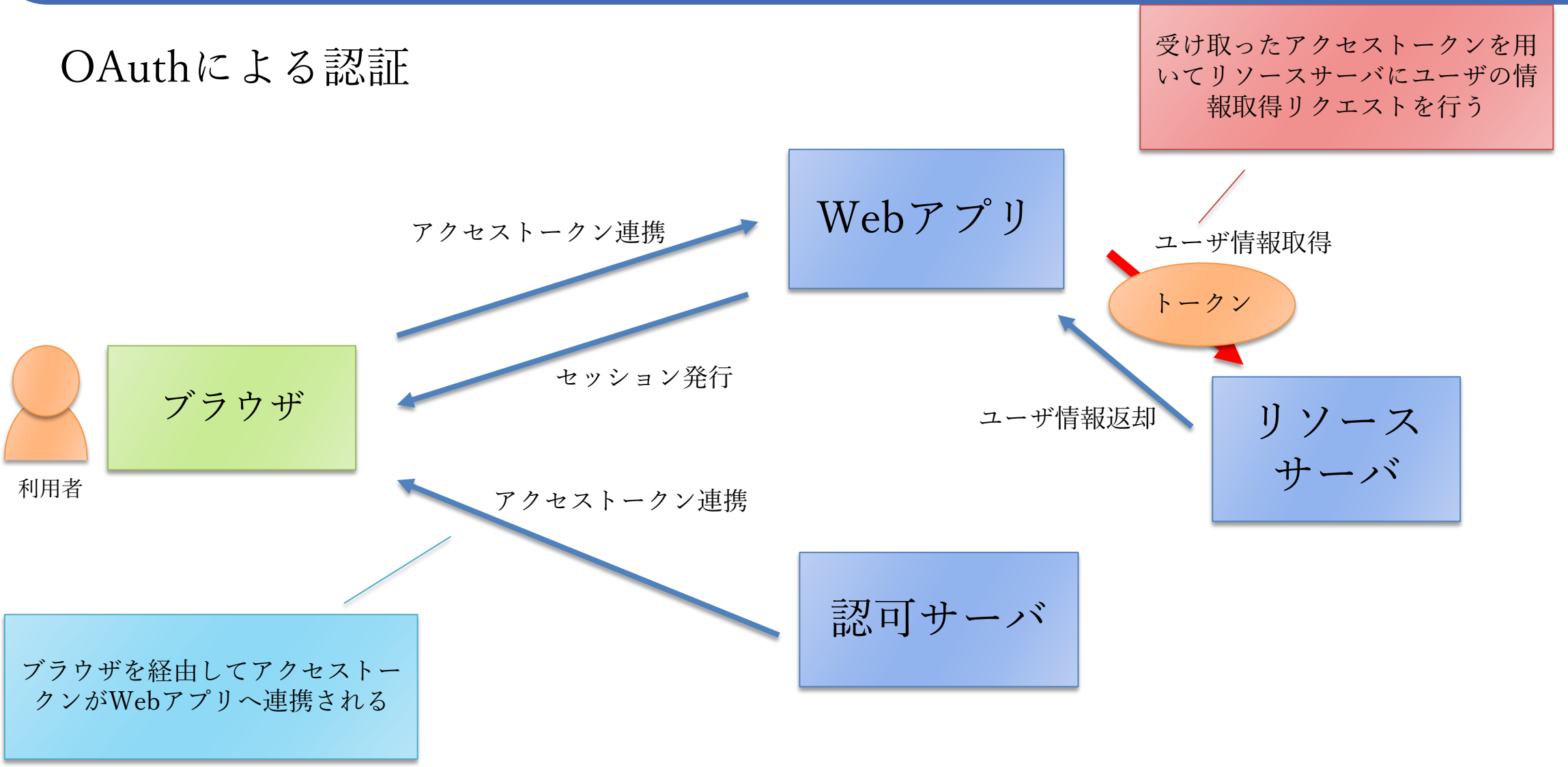
標準技術選定を誤った例

OAuthによる認証（ImplicitGrant）



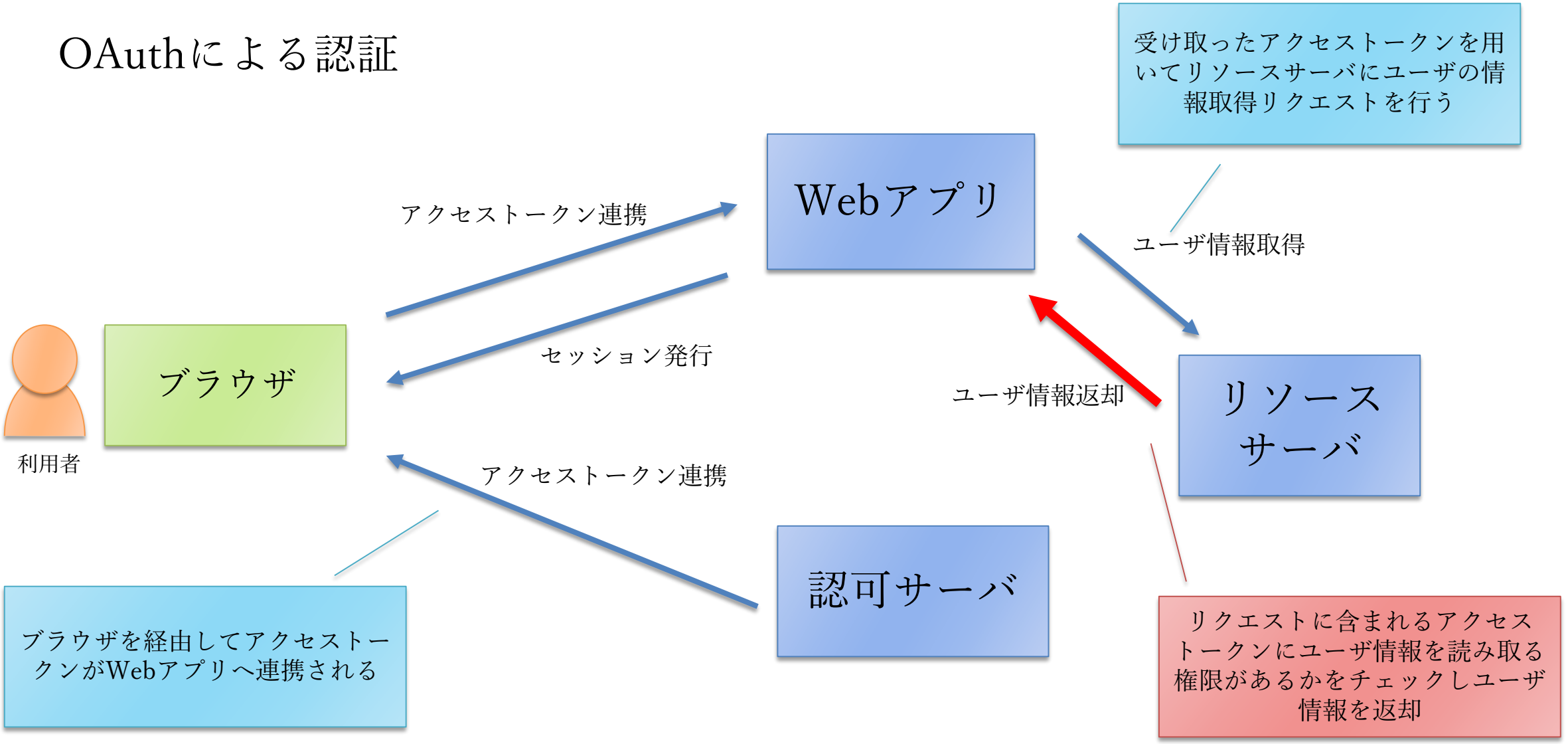
標準技術選定を誤った例

OAuthによる認証



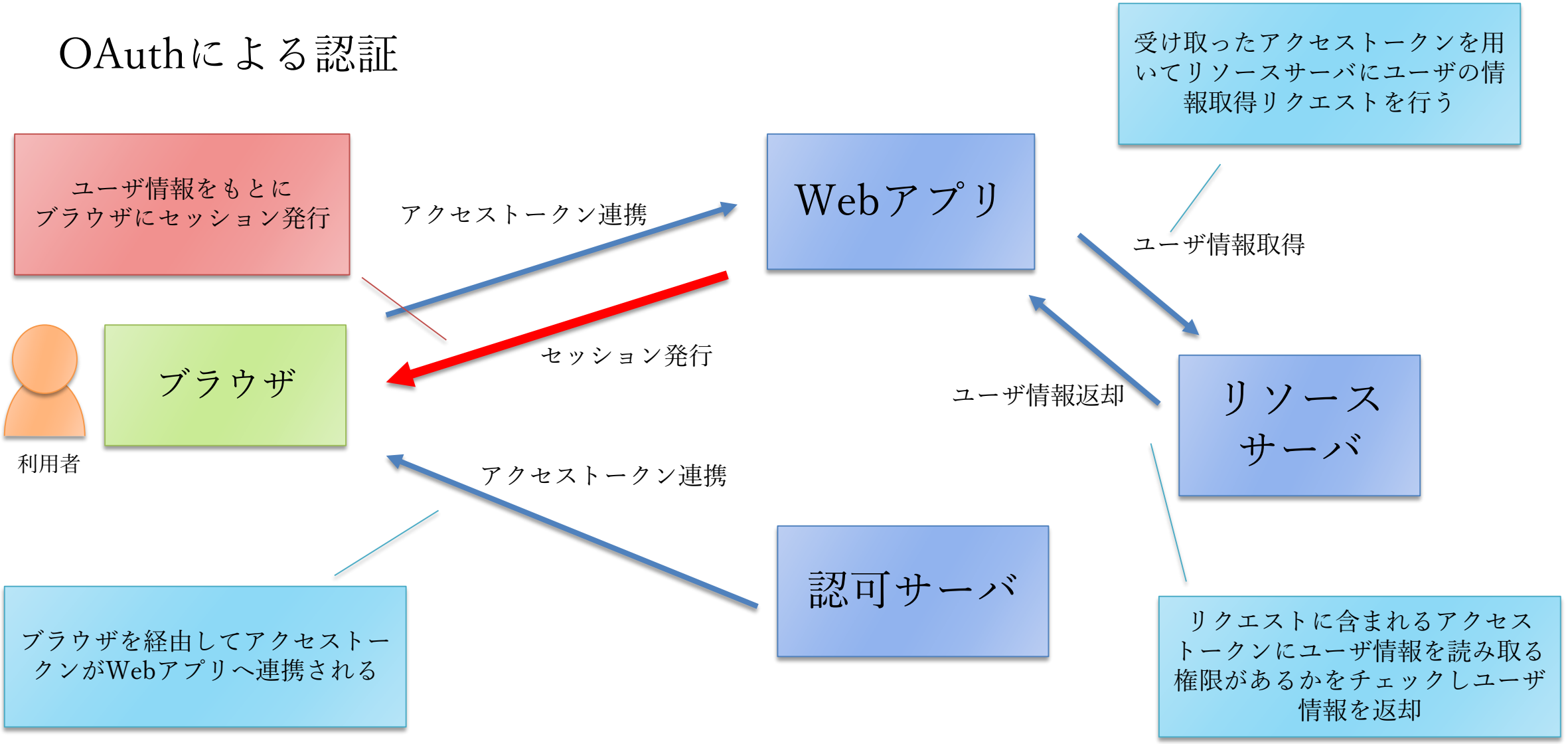
標準技術選定を誤った例

OAuthによる認証



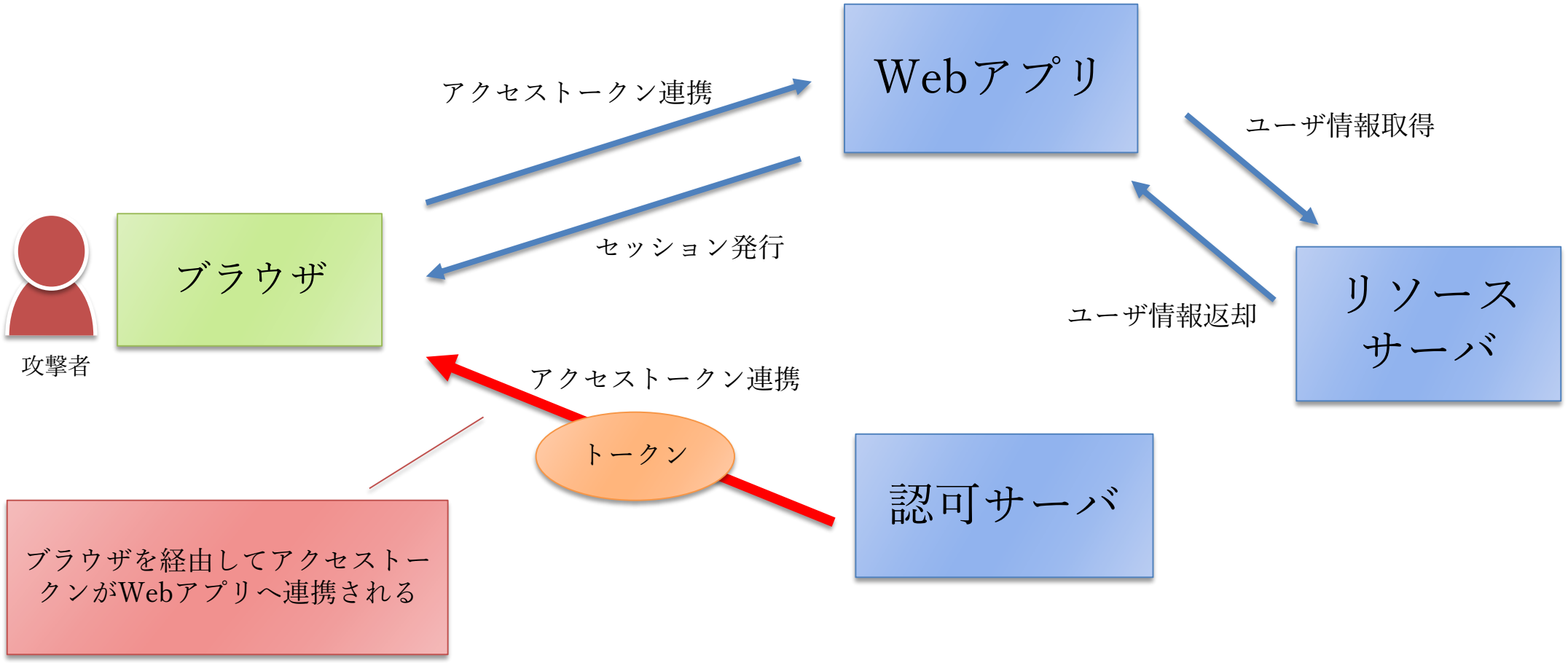
標準技術選定を誤った例

OAuthによる認証



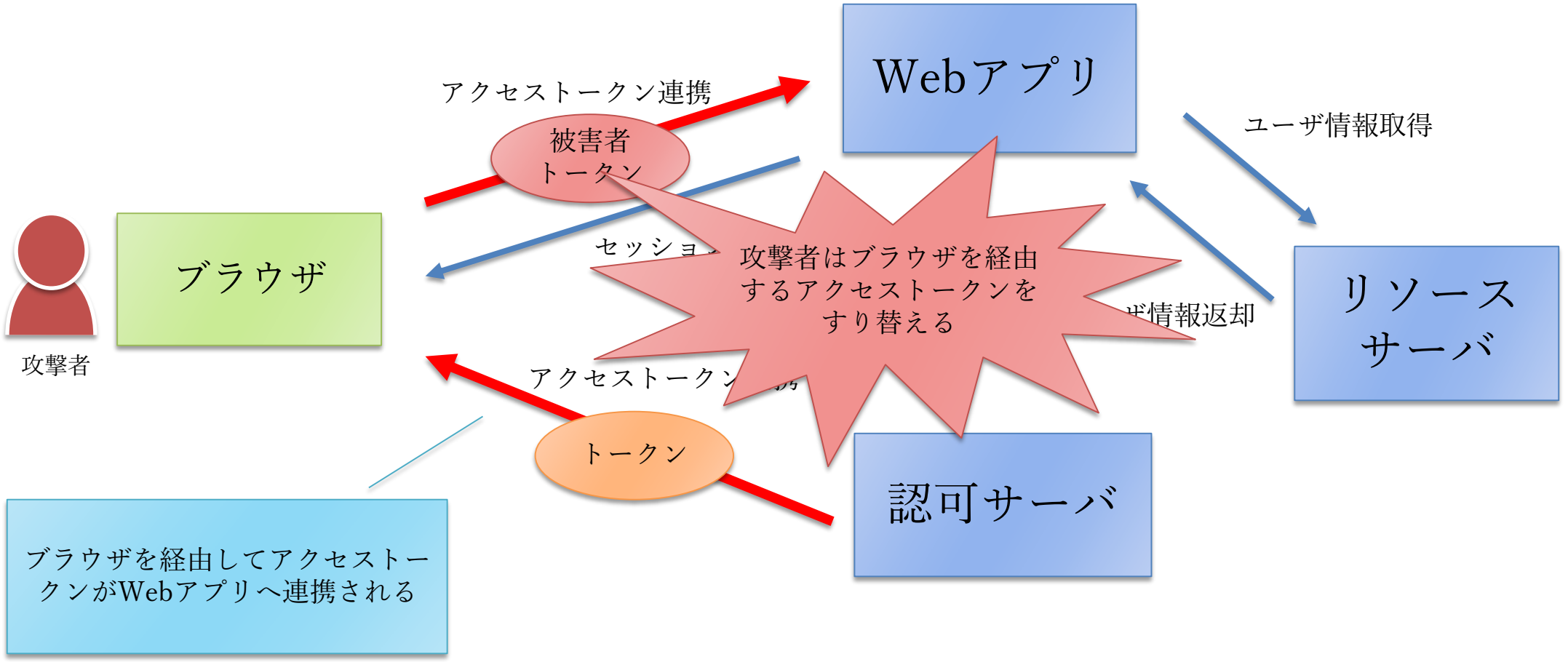
OAuth認証による脆弱性

攻撃者によるなりすまし



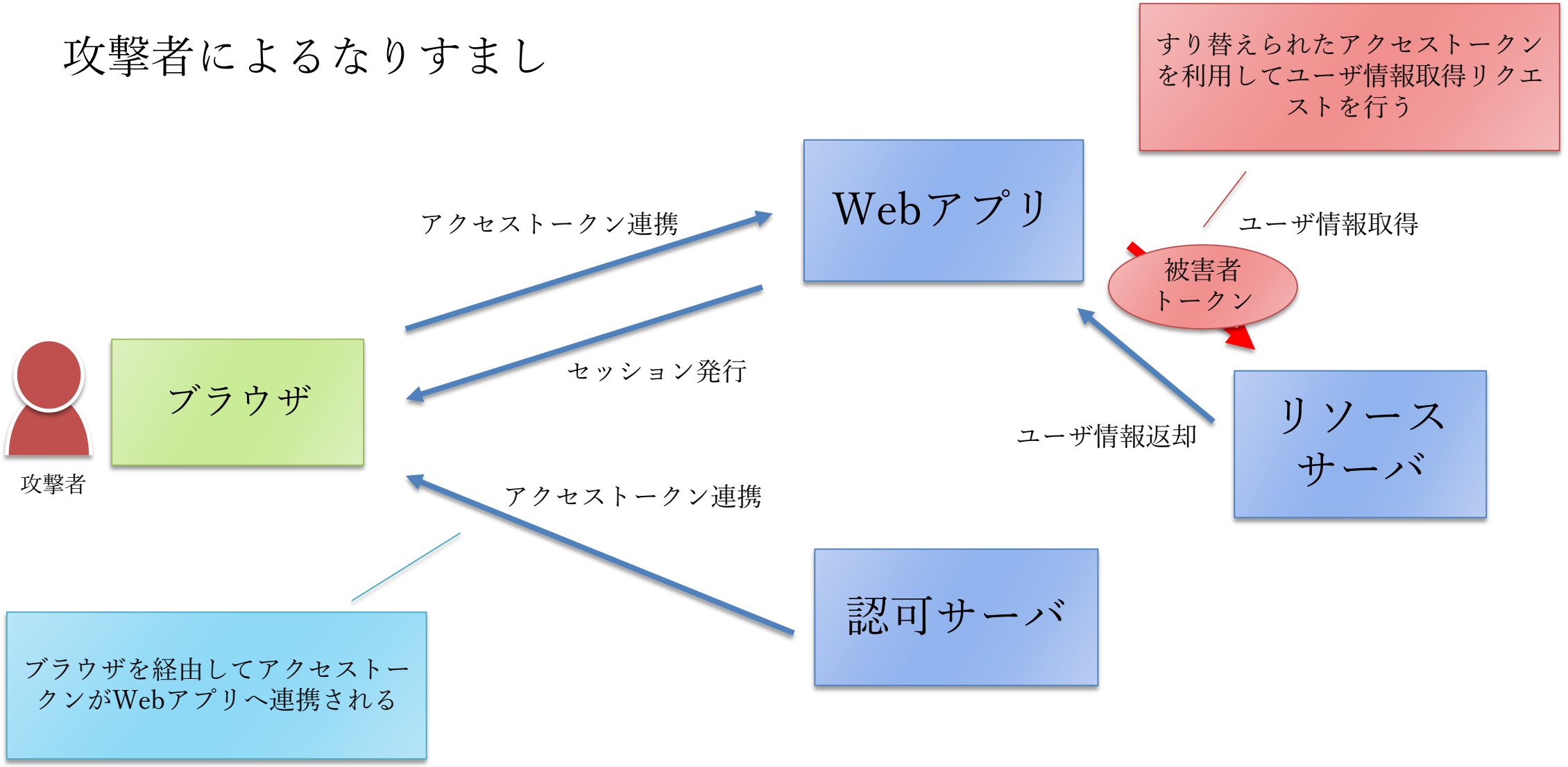
OAuth認証による脆弱性

攻撃者によるなりすまし



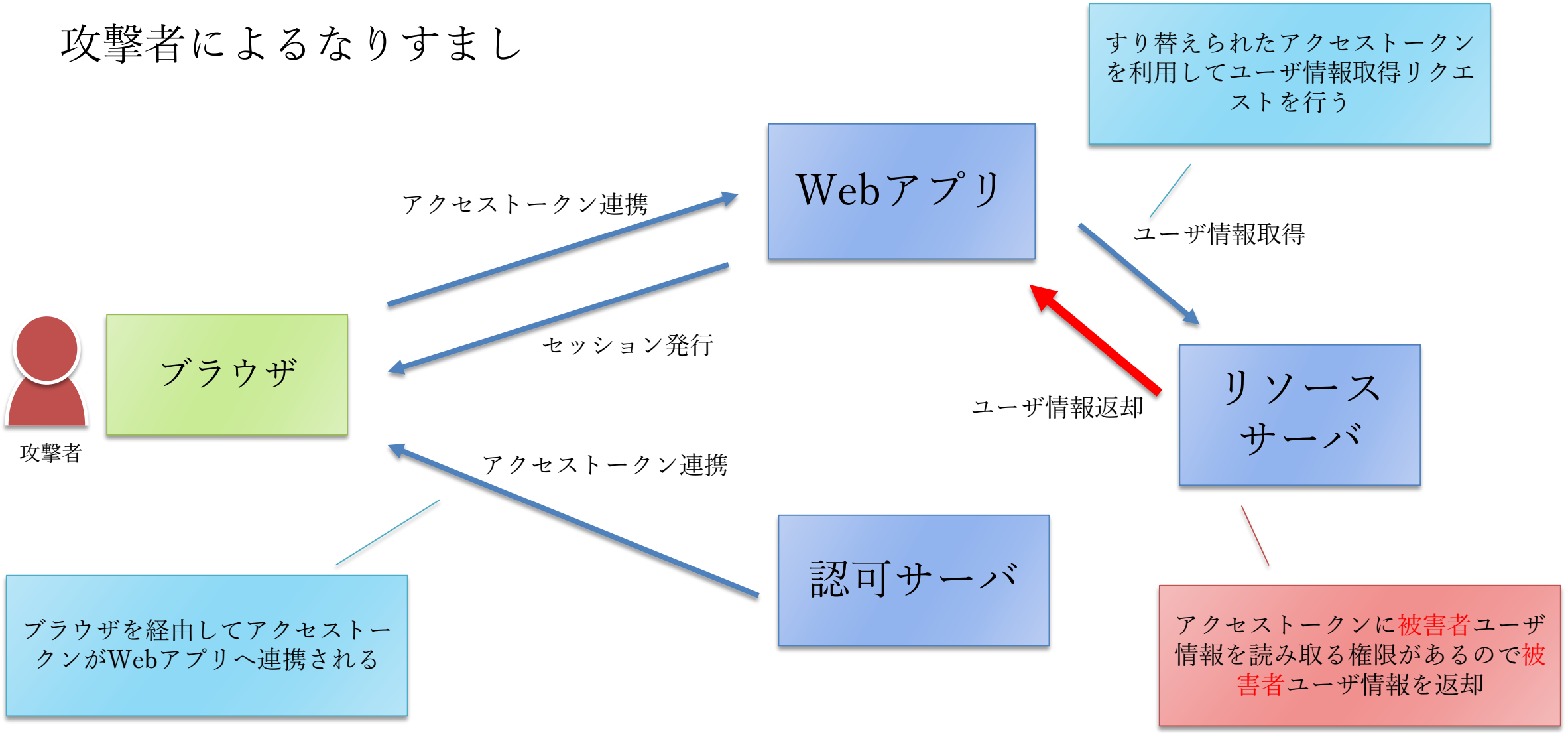
OAuth認証による脆弱性

攻撃者によるなりすまし



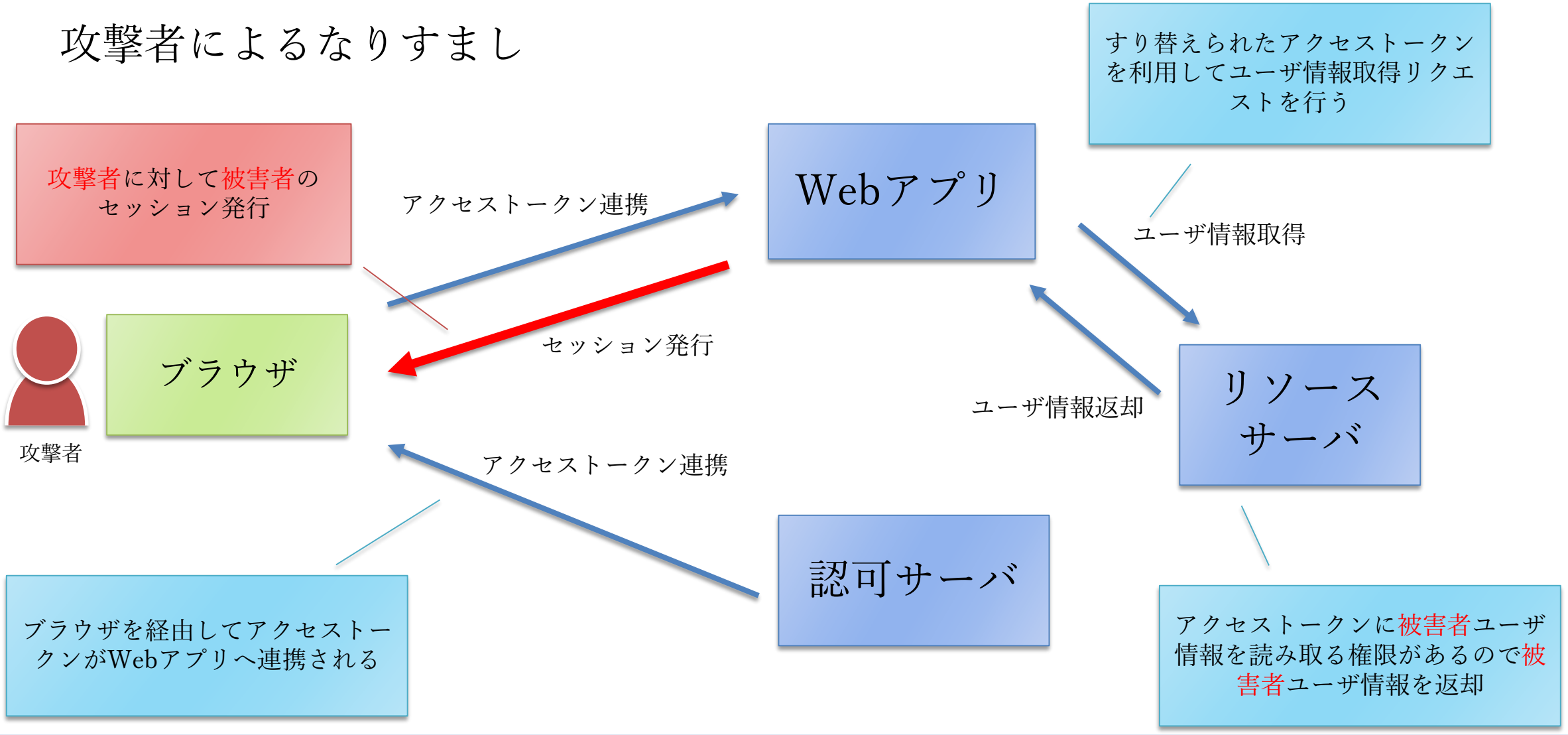
OAuth認証による脆弱性

攻撃者によるなりすまし



OAuth認証による脆弱性

攻撃者によるなりすまし



脆弱性を防ぐためには

- **用途に沿った標準技術を選択していれば脆弱性を防ぐことができた**
  - 認証情報の連携に利用すべき代表的な標準技術
    - OpenID Connect
    - SAML
    - etc…

先ほどの例をOpenID Connectに置き換えて考えてみる

OpenID Connectとは

- **IDトークンを用いて認証情報を連携する仕様**
  - IDトークンとはユーザの認証情報をJWT（Json Web Token）形式にしたトークン
  - IDトークン自身に署名が施されており改ざんすることができない

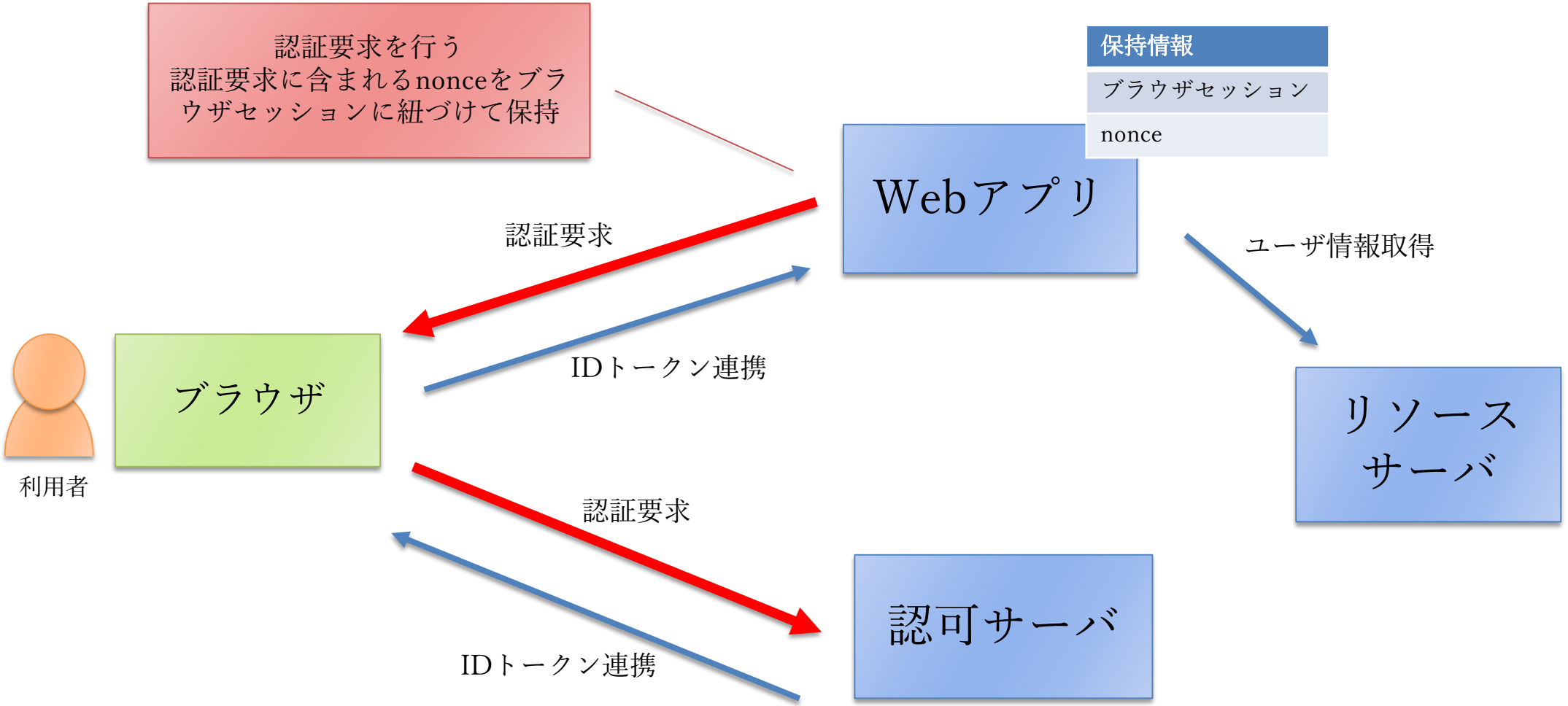
- **IDトークンには以下のような情報が紐づいている（一部抜粋）**
  - sub（ユーザ識別子）
  - aud（発行対象）
  - iss（発行元）
  - iat（発行日時）
  - acr（認証レベル）
  - auth_time（認証日時）
  - nonce（リプレイアタック防止パラメータ）

アクセストークンと同様

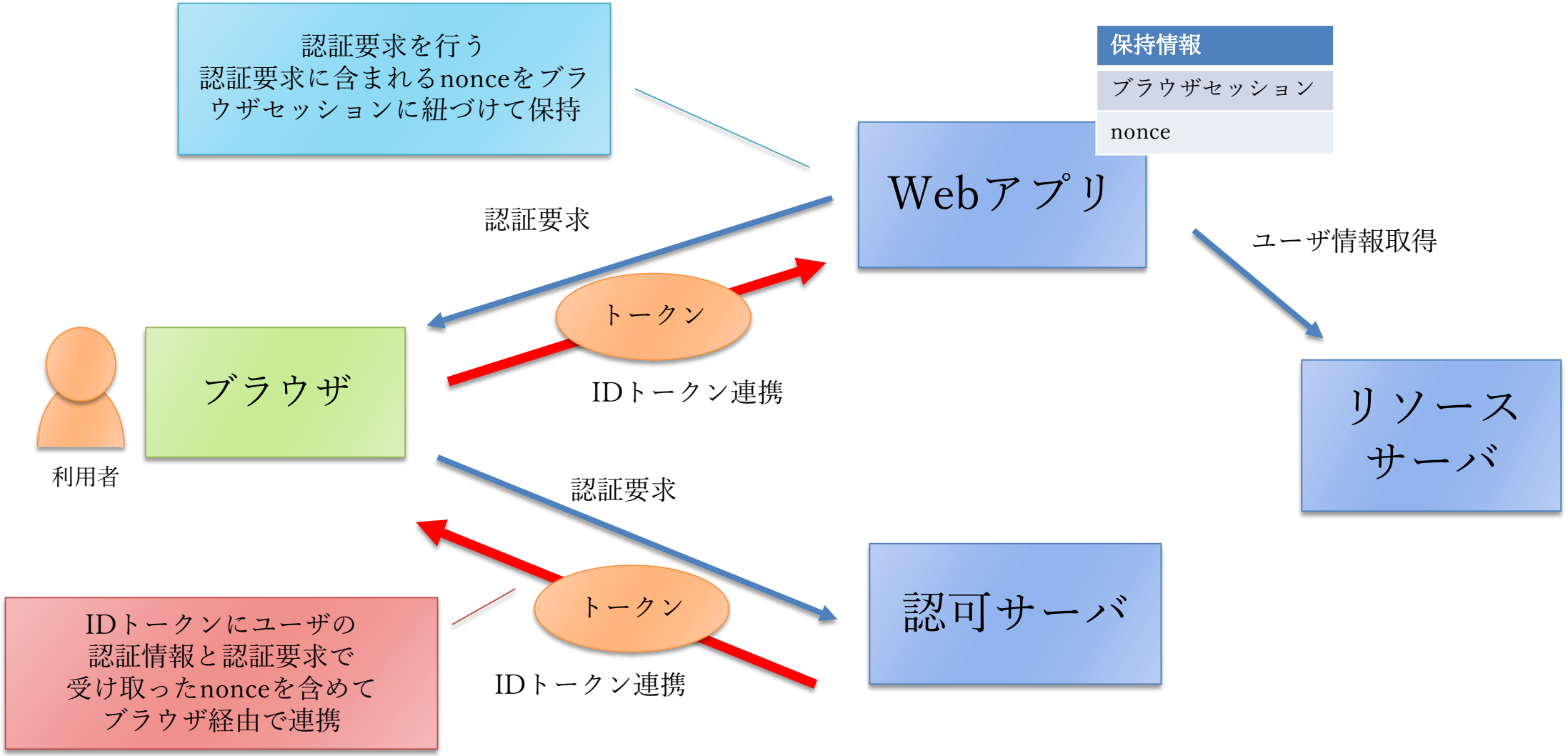
IDトークン独自

- **連携先アプリケーション(Relying Pary)は署名されたIDトークンを検証し、認証情報を読み取ることで認証連携を実現する**
  - 必要に応じ、IDトークンとともに連携されるアクセストークンを利用して適宜リソースサーバから情報取得

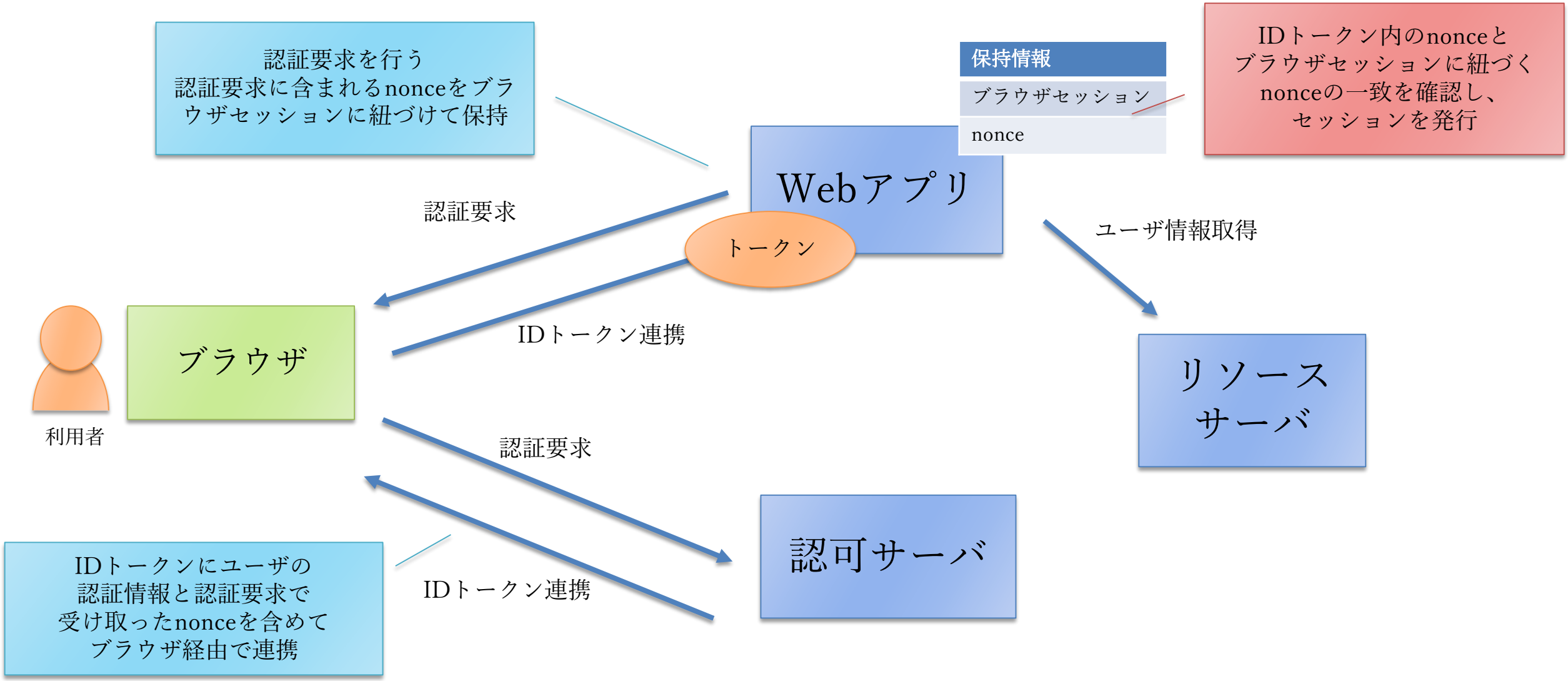
OpenID Connectを利用した場合



OpenID Connectを利用した場合

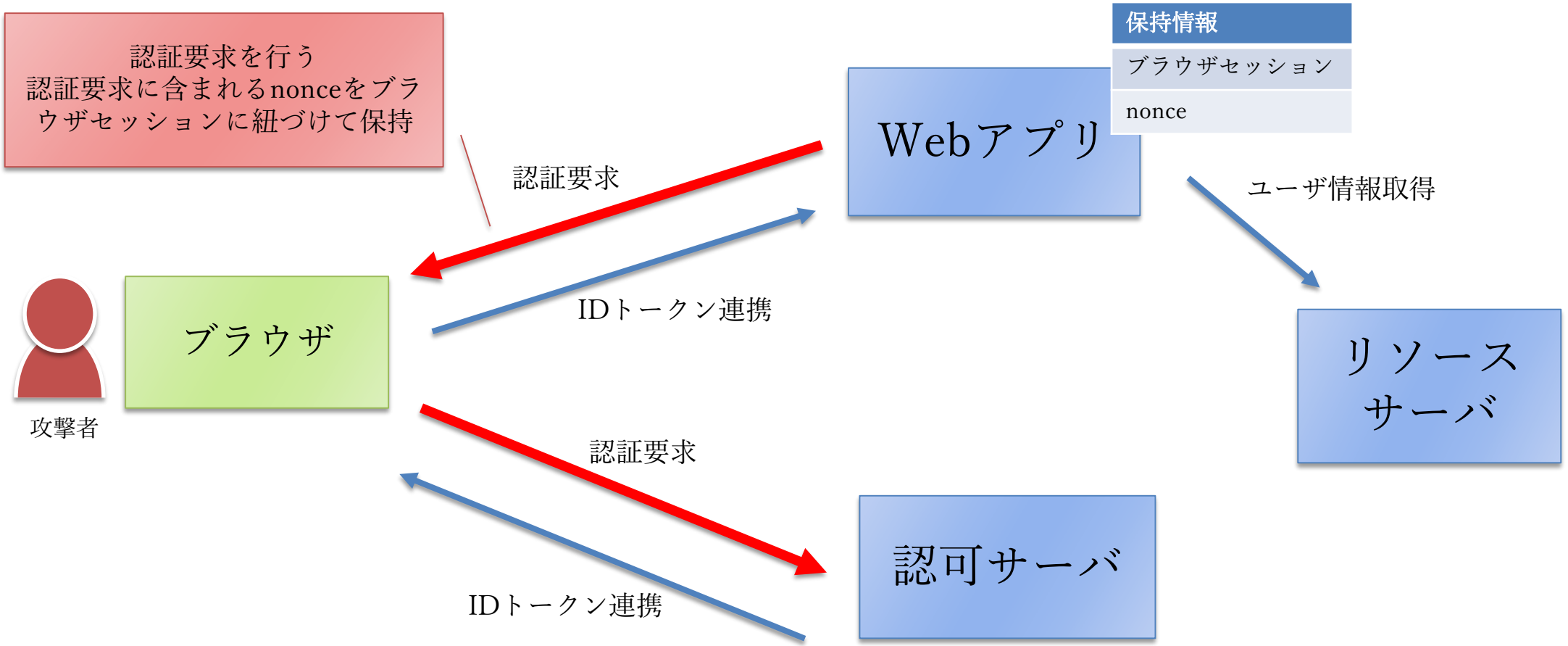


OpenID Connectを利用した場合



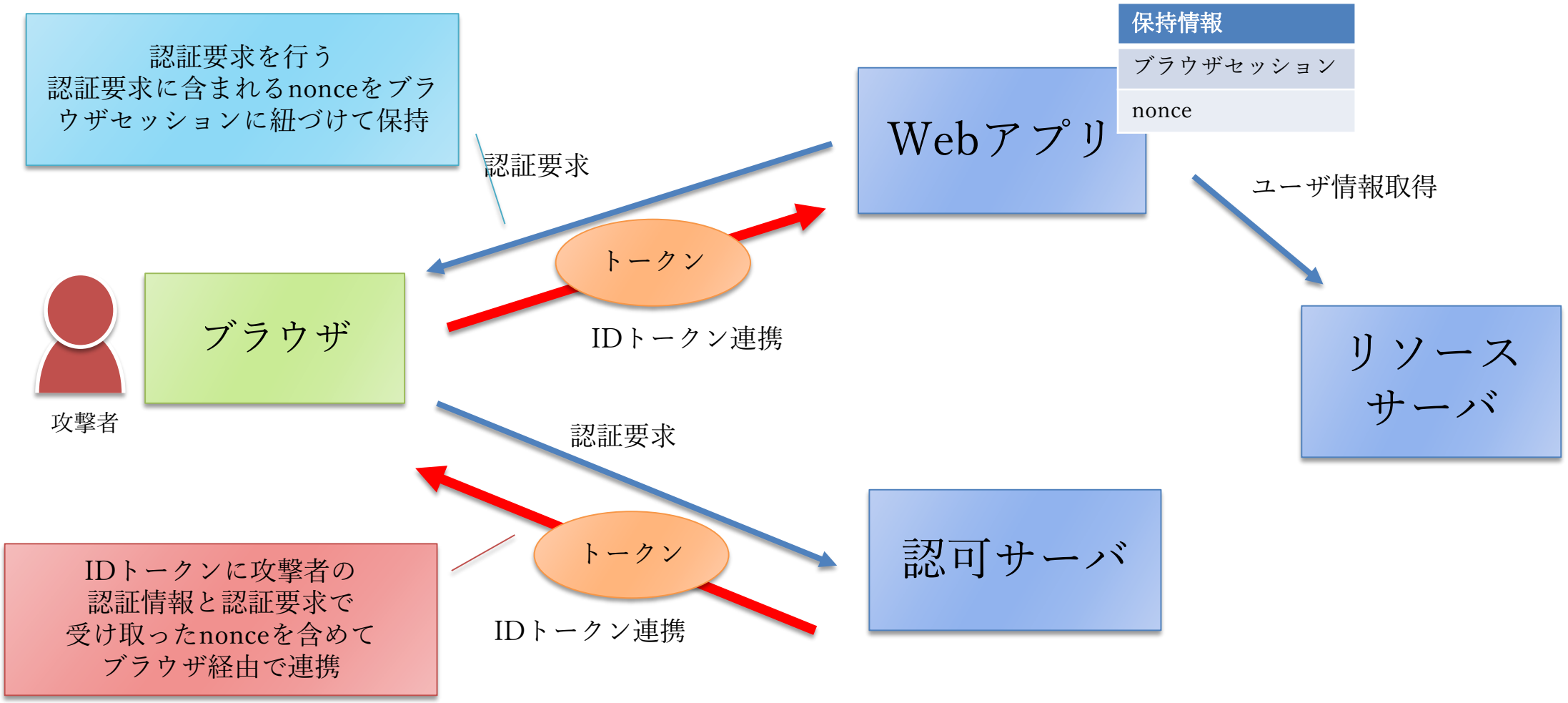
IDトークンをすり替えた場合

OpenID Connectで連携されるトークンをすり替える



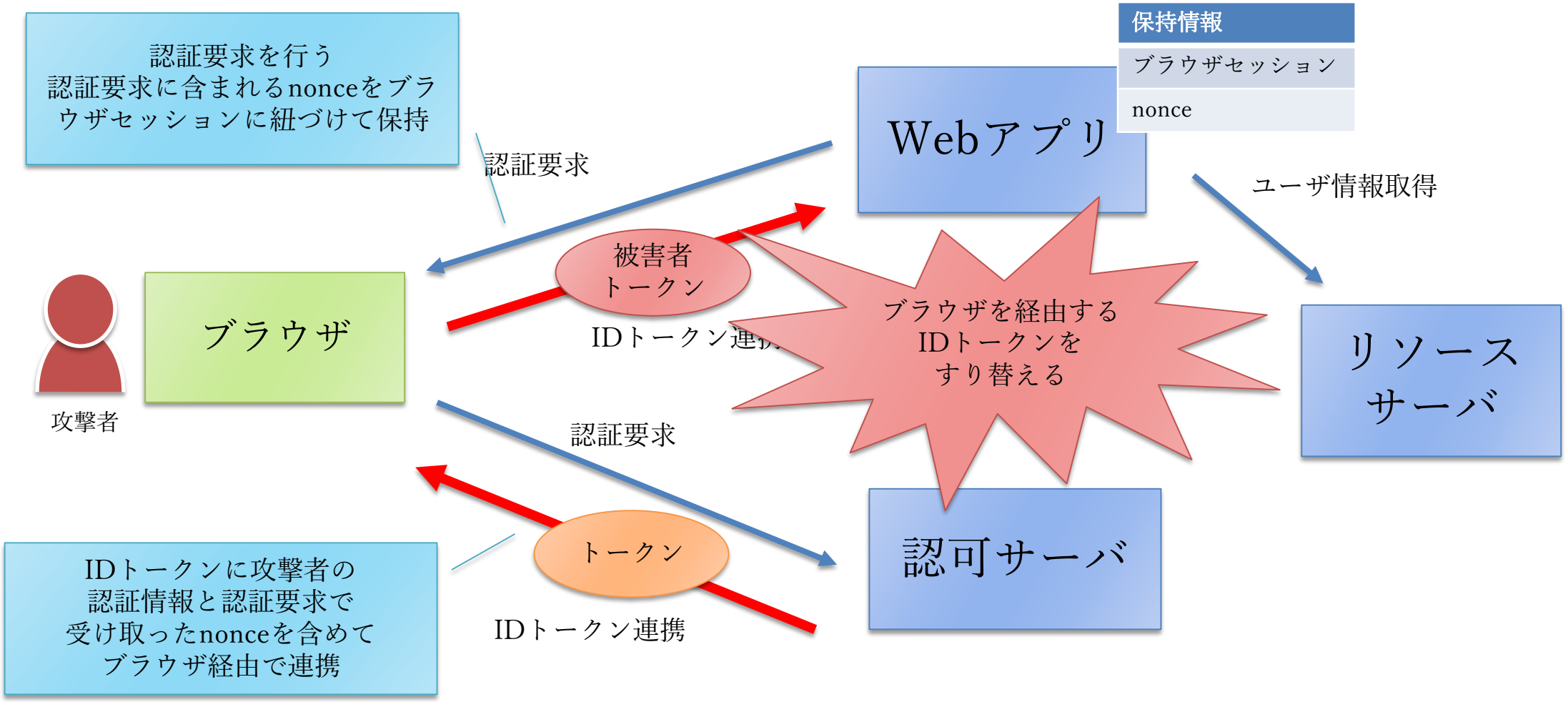
IDトークンをすり替えた場合

OpenID Connectで連携されるトークンをすり替える



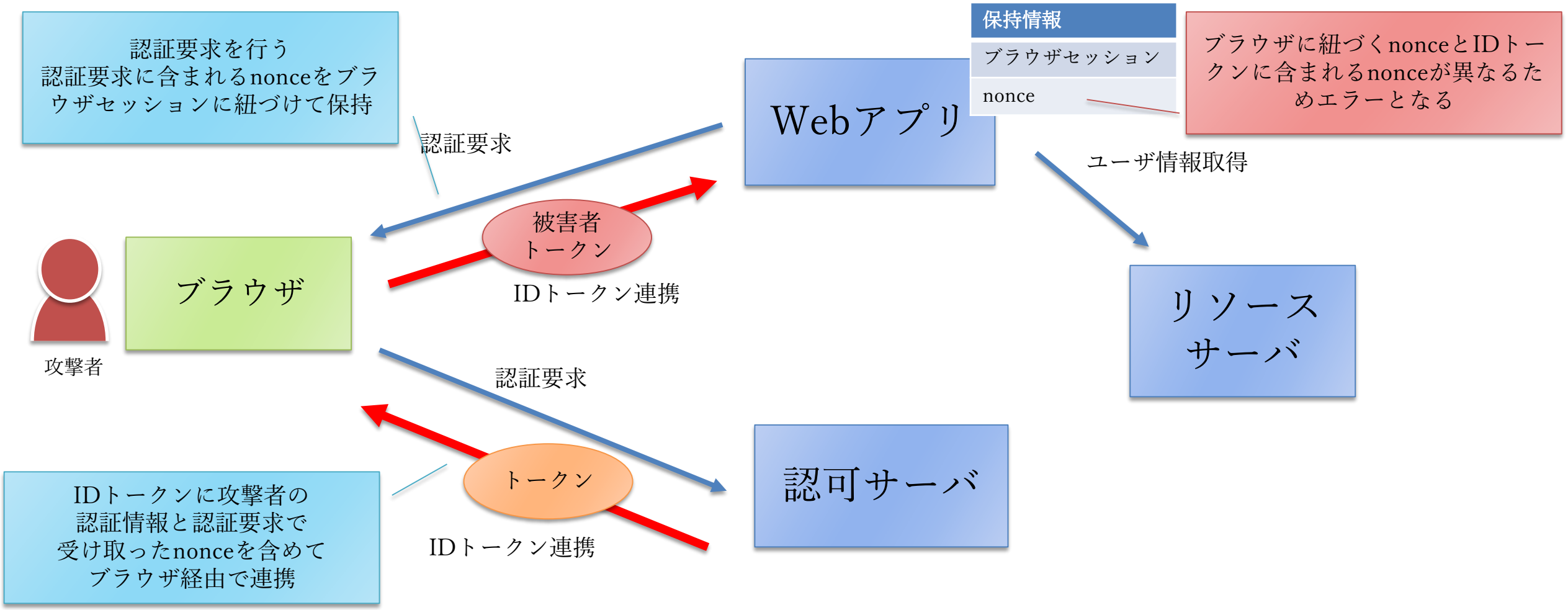
IDトークンをすり替えた場合

OpenID Connectで連携されるトークンをすり替える



IDトークンをすり替えた場合

OpenID Connectで連携されるトークンをすり替える



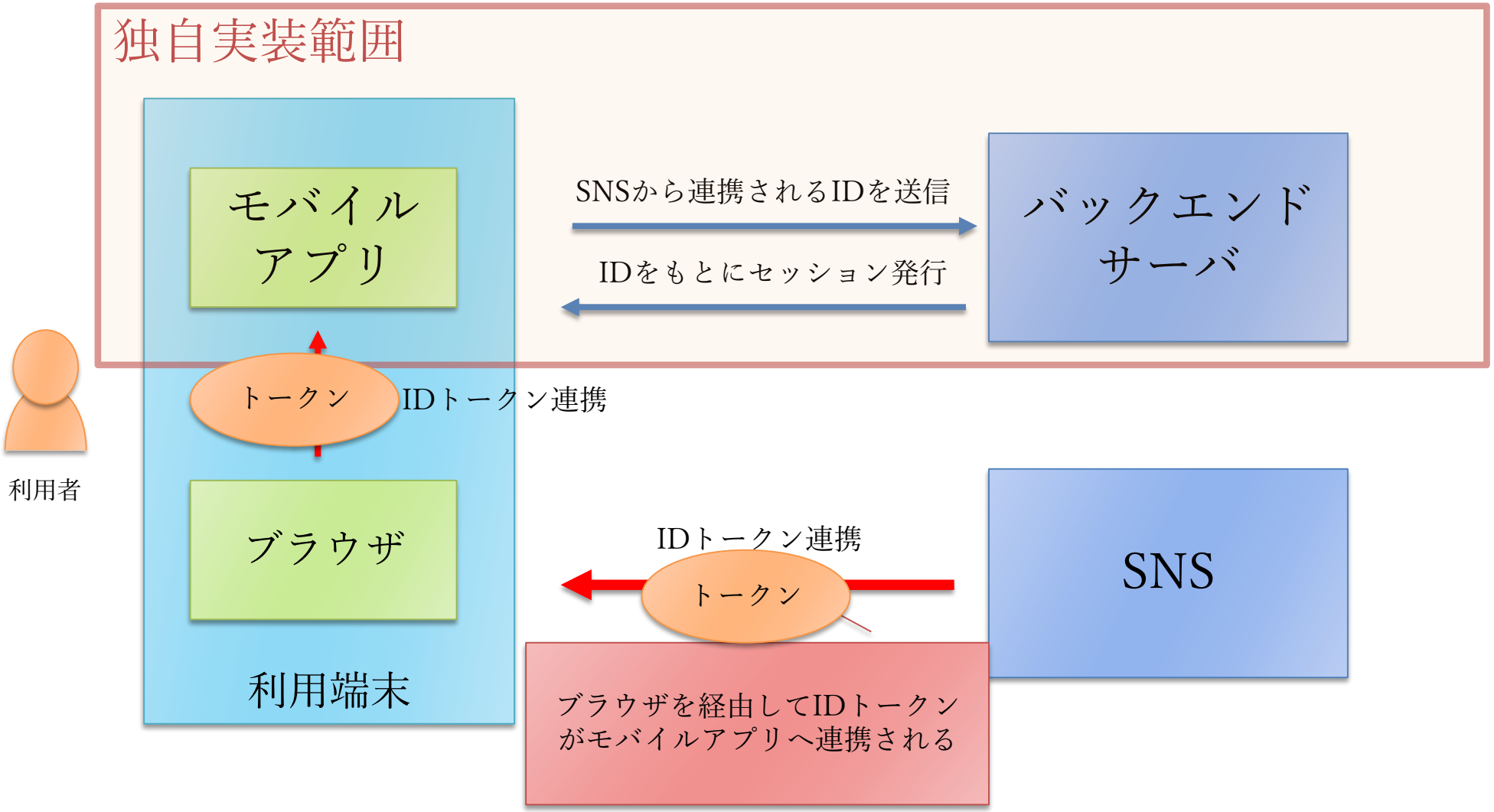
用途に沿っていても適用方法を誤ると効果はない

標準技術の適用方法を誤った例

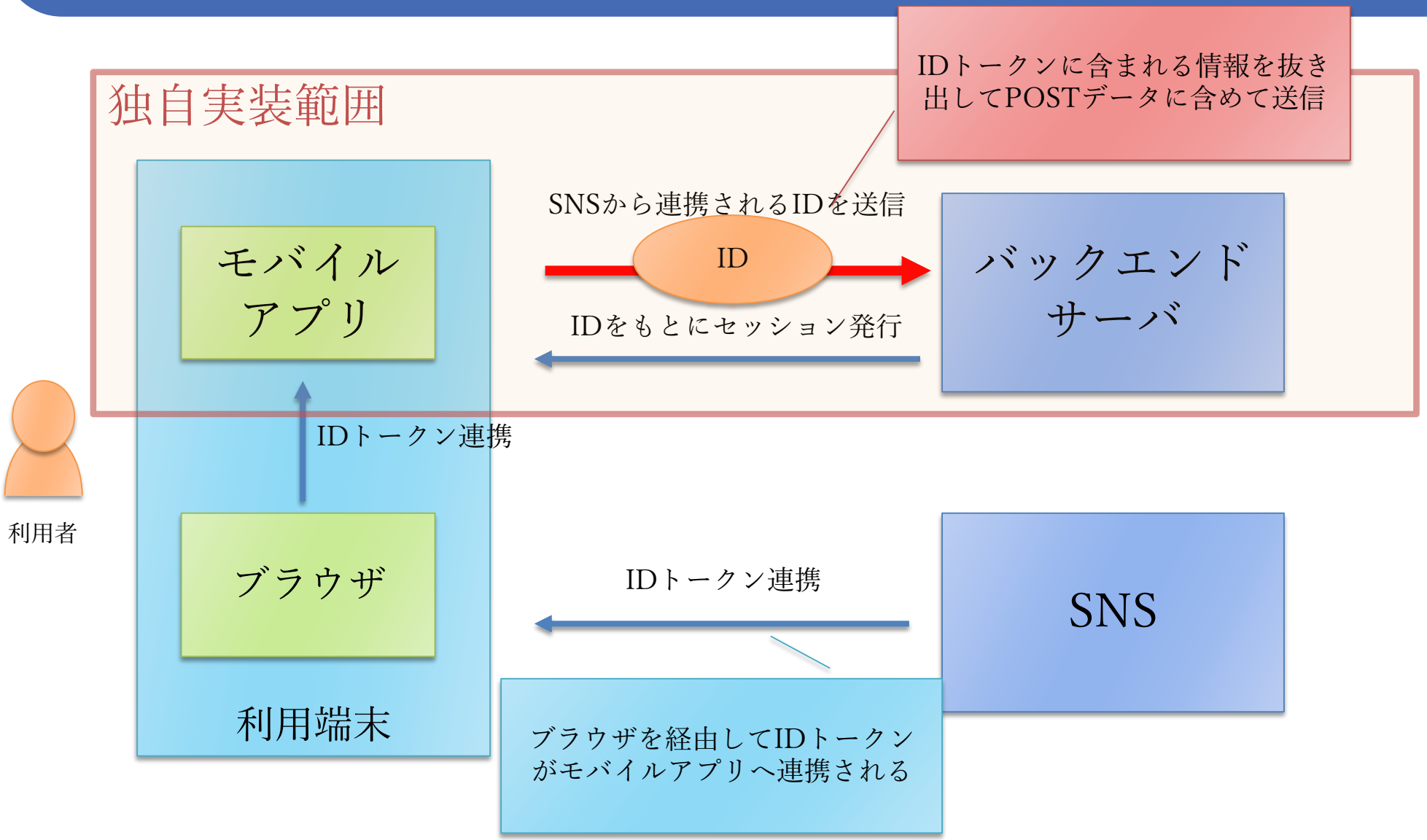
ロ 例：あるモバイルアプリケーションの認証機能

- バックエンドサーバからモバイルアプリへのセッション発行にSNSアカウントを用いた外部ID連携機能を実装
- モバイルアプリとSNSの認証情報連携にOpenID Connectを利用して安全に認証情報の連携を行うように設計
- しかし、モバイルアプリとバックエンドサーバ間のセッション発行は独自実装を用いたことにより脆弱性が発生した

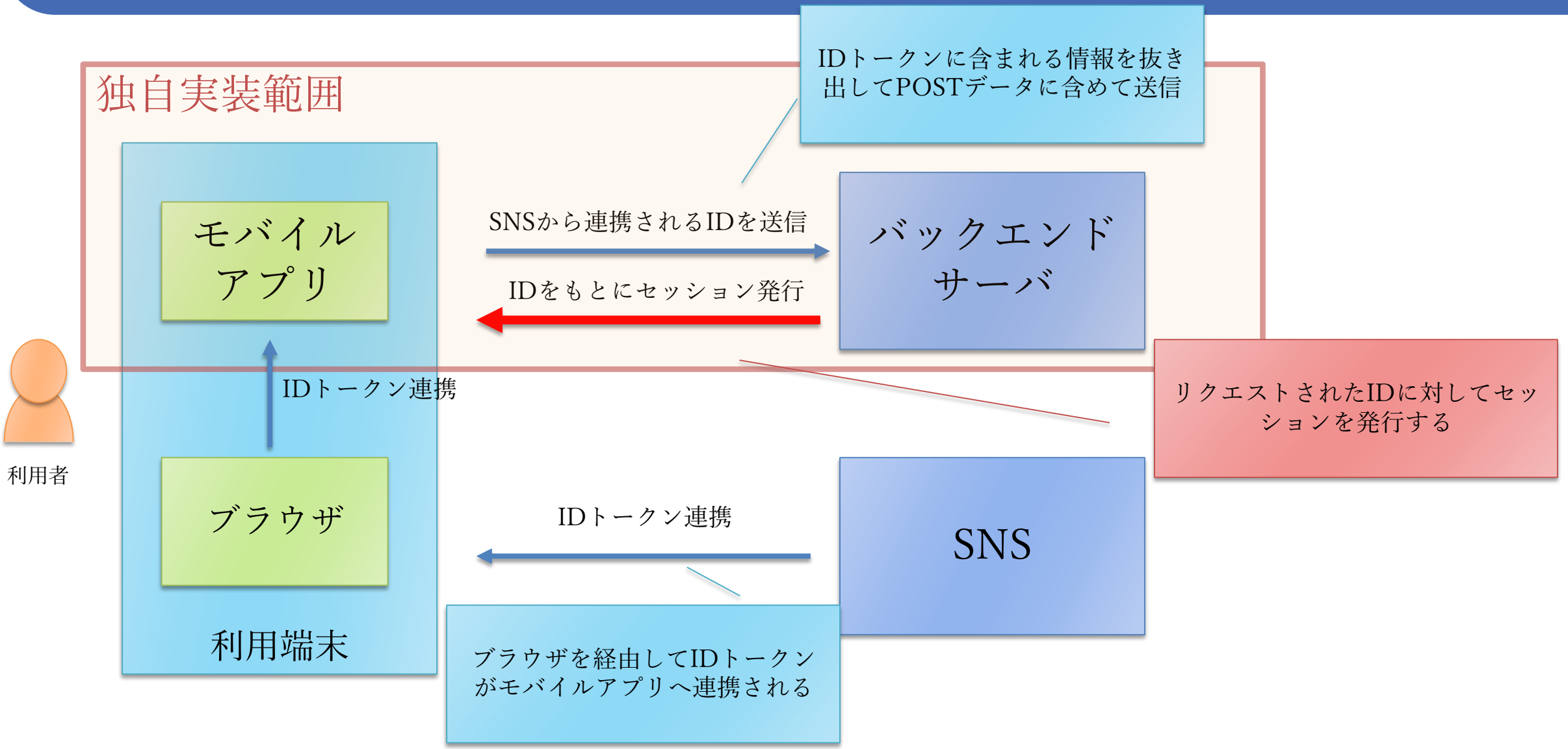
標準技術の適用方法を誤った例



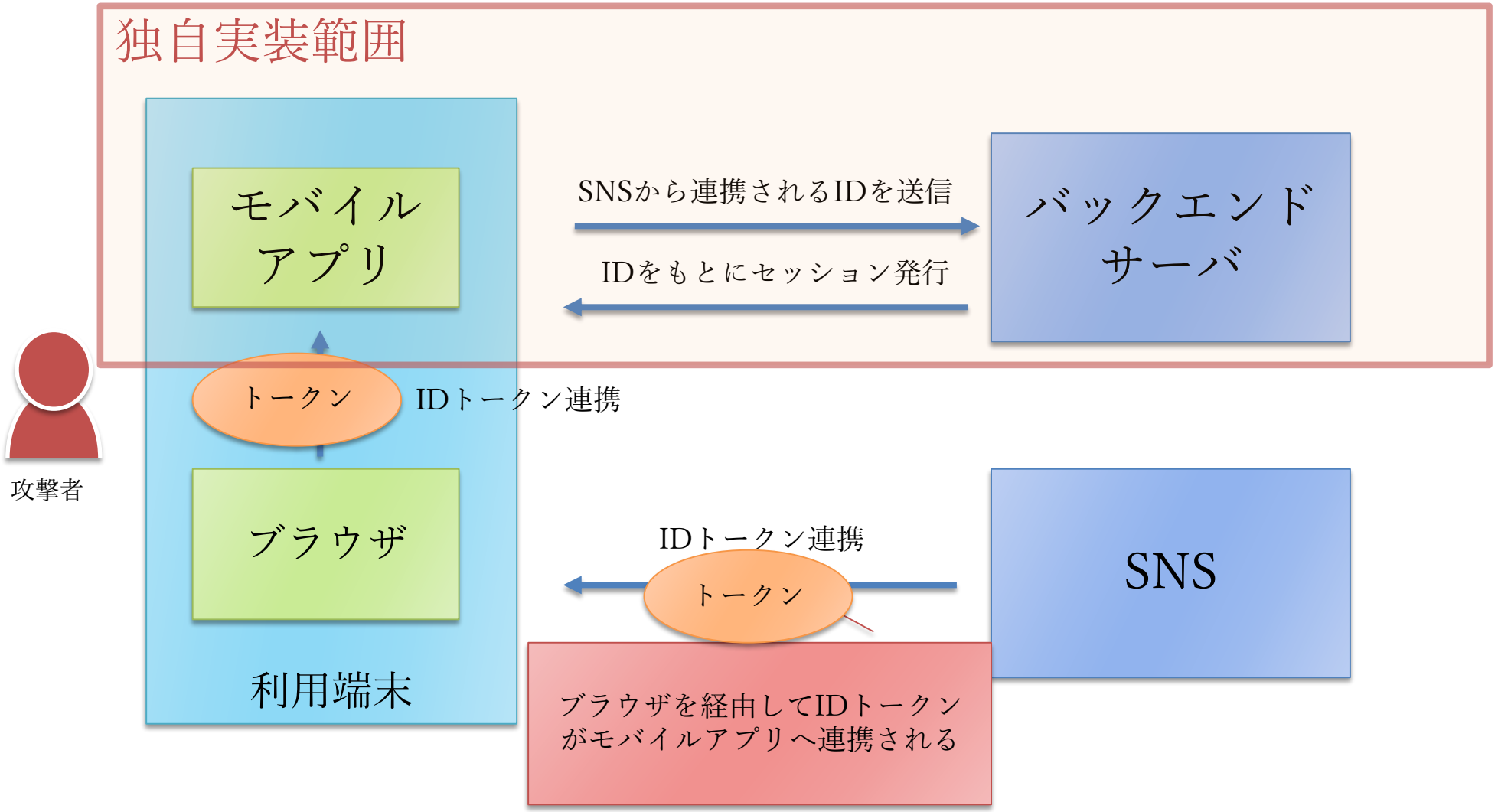
標準技術の適用方法を誤った例



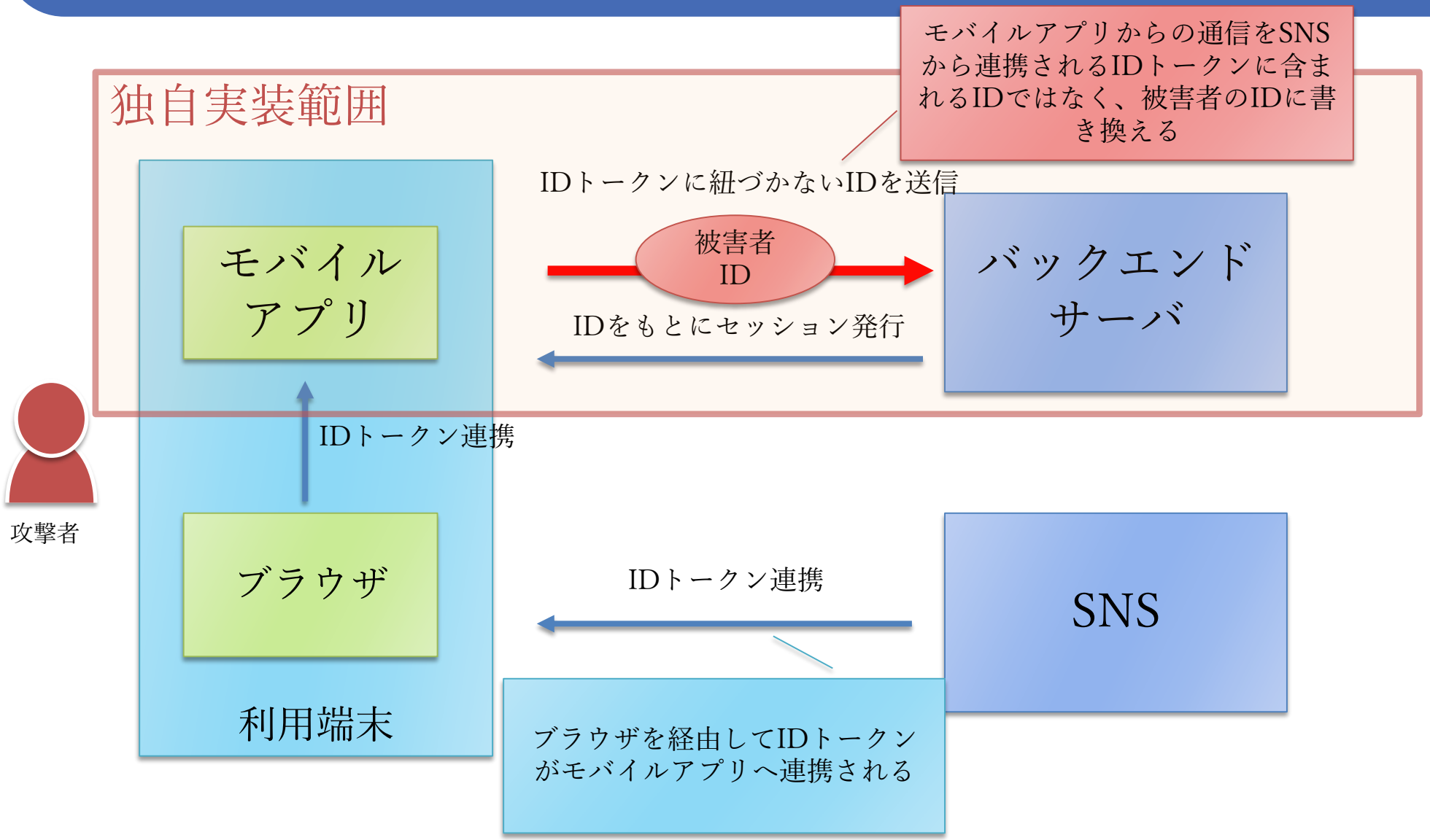
標準技術の適用方法を誤った例



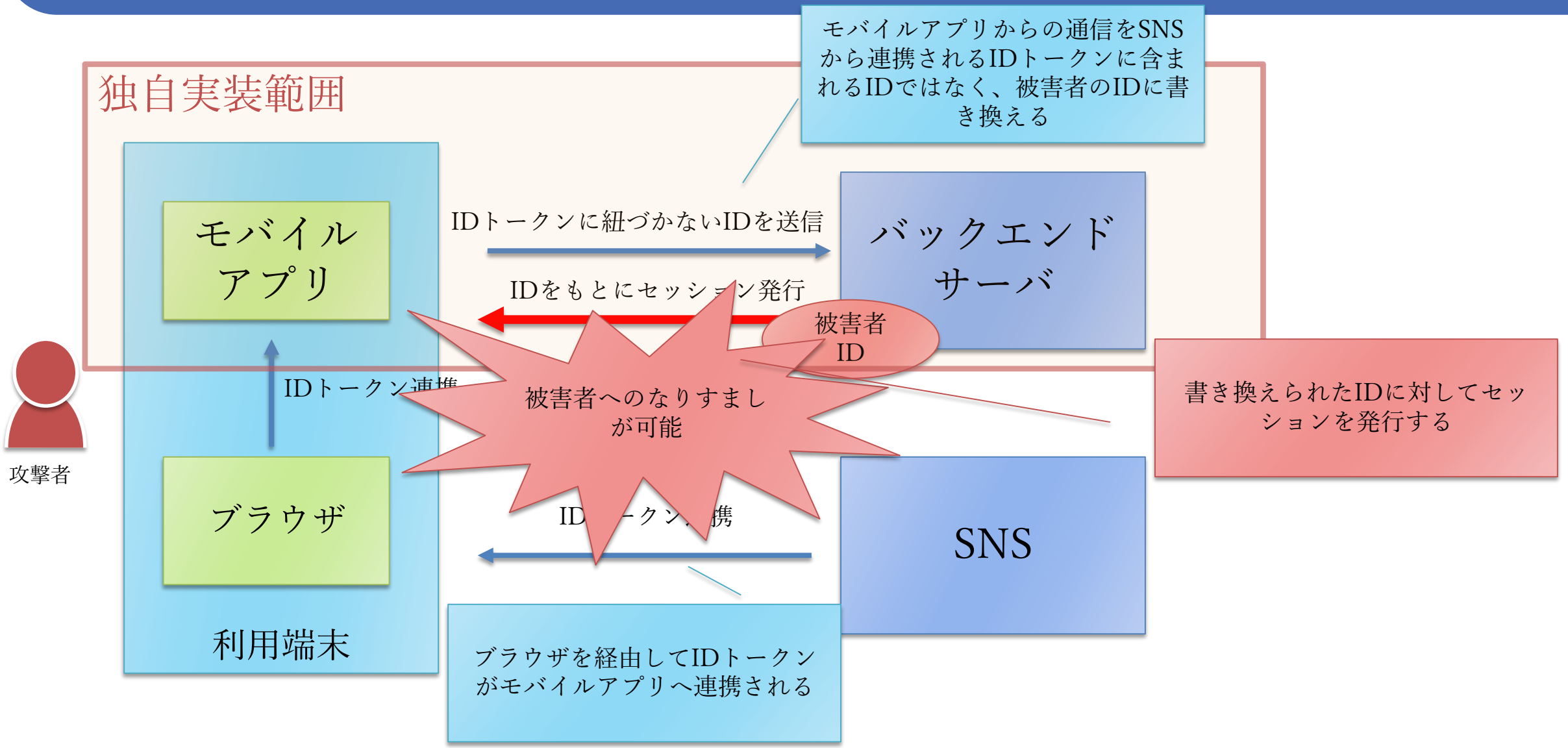
脆弱性を突く攻撃手法


独自実装範囲
モバイル
アプリ
SNSから連携されるIDを送信
IDをもとにセッション発行
バックエンド
サーバ
トークン
IDトークン連携
攻撃者
ブラウザ
IDトークン連携
トークン
SNS
利用端末
ブラウザを経由してIDトークン
がモバイルアプリへ連携される

脆弱性を突く攻撃手法

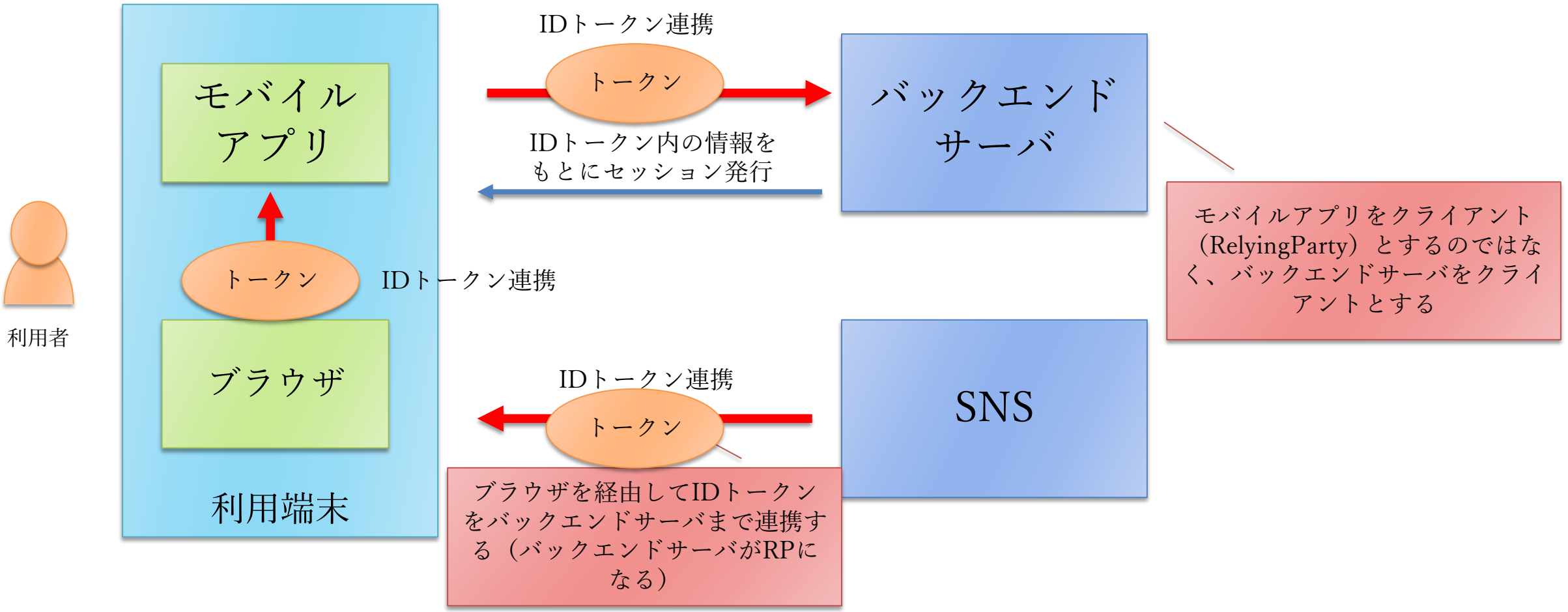


脆弱性を突く攻撃手法



脆弱性を生まない構成

認証情報を必要としているシステムが認証要求と認証情報の検証を行う



ここまでのまとめ

ロ システムを構築するうえで標準技術を知ることが大切である

- どういうときに利用するのか
  - 仕様にはユースケースが記載されていることが多く、ユースケースにマッチした仕様を選択する必要がある
- 正しい適用範囲は何か
  - 登場人物（システム）が何であり、どの部分へ適用すべき仕様であるか
- それによってどの部分が担保されるのか
  - 利用しようとする標準技術が守ってくれない部分は別の技術等により守る必要がある

認証技術動向の紹介

FIDO2

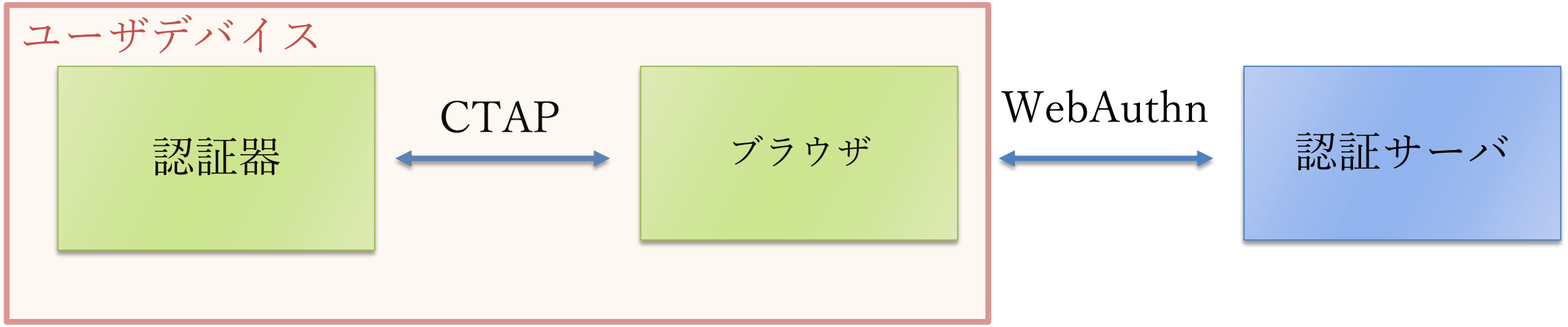
FIDO2とは

- パスワードレスを目指すFIDO Allianceが策定する認証技術
- FIDOではUAFやU2Fといったプロトコルに分かれていたがFIDO2で統合し、ブラウザからFIDOを利用するためのWebAuthnおよび認証器との通信を規定するCTAPによって主に構成される仕様
- 公開鍵暗号による認証を用いる
- 多くの企業・団体がサポートに向けて動いている
  - Androidが2019年2月にFIDO2認定を取得
  - Windows Helloも2019年5月にFIDO2認定取得
  - 2019年3月にW3C（Web技術の標準化を行う団体）によってWeb標準となっており、主要ブラウザがサポート

FIDO2の構成要素（簡易）

ロ FIDO2の仕様

- ブラウザ・デバイスから認証器への通信はCTAP
- ブラウザから認証サーバへの通信はWebAuthn



認証器とは

- 公開鍵認証を実施するために秘密鍵をもとに署名の生成を行う機器
- PINや生体認証によるユーザ認証機能を持つものもある
- 例：
  - デバイス内に内蔵している場合
    - Windows Helloで利用されるPC内蔵認証器
    - Android（スマートフォン）内蔵認証器
  - 外部接続する場合
    - USB接続
    - BLE（Bluetooth）接続

出典：Google
[https://cloud.google.com/titan-security-key/]

FIDO2のメリット

- **公開鍵認証を利用するため、認証サーバ側でパスワードや生体情報を保持する必要がないため漏洩の危険性が低い**
  - 認証サーバでは公開鍵のみを保持しており、万が一公開鍵が漏洩した場合にも不正認証はおこなえない
- **WebAuthnによってブラウザ（アプリ）からJavaScriptのAPIを呼び出すことでFIDO利用が容易となった**
- **公開鍵認証＋認証器側で生体やPINによる認証を行うことで簡単に多要素認証を実施できる**

認証要素とは

□ 認証要素とは以下の三種類で構成される

- What You Know（知っている）
  - ユーザが記憶している情報による認証
    - パスワード、生年月日、PIN、秘密の質問…
  - ユーザの記憶できる範囲であることが多く推測・漏洩のリスクが高い
- What You Have（持っている）
  - ユーザが保持しているものによる認証
    - OTP、ICカード、クライアント証明書、公開鍵認証…
  - 推測による不正認証は難しいが、紛失時のリスクが高い
- What You Are（あなたである）
  - ユーザ自身の情報による認証
    - 生体認証
  - 利用する機器により認証精度が異なり、他人受け入れ・本人拒否の可能性がある

多要素認証と多段階認証

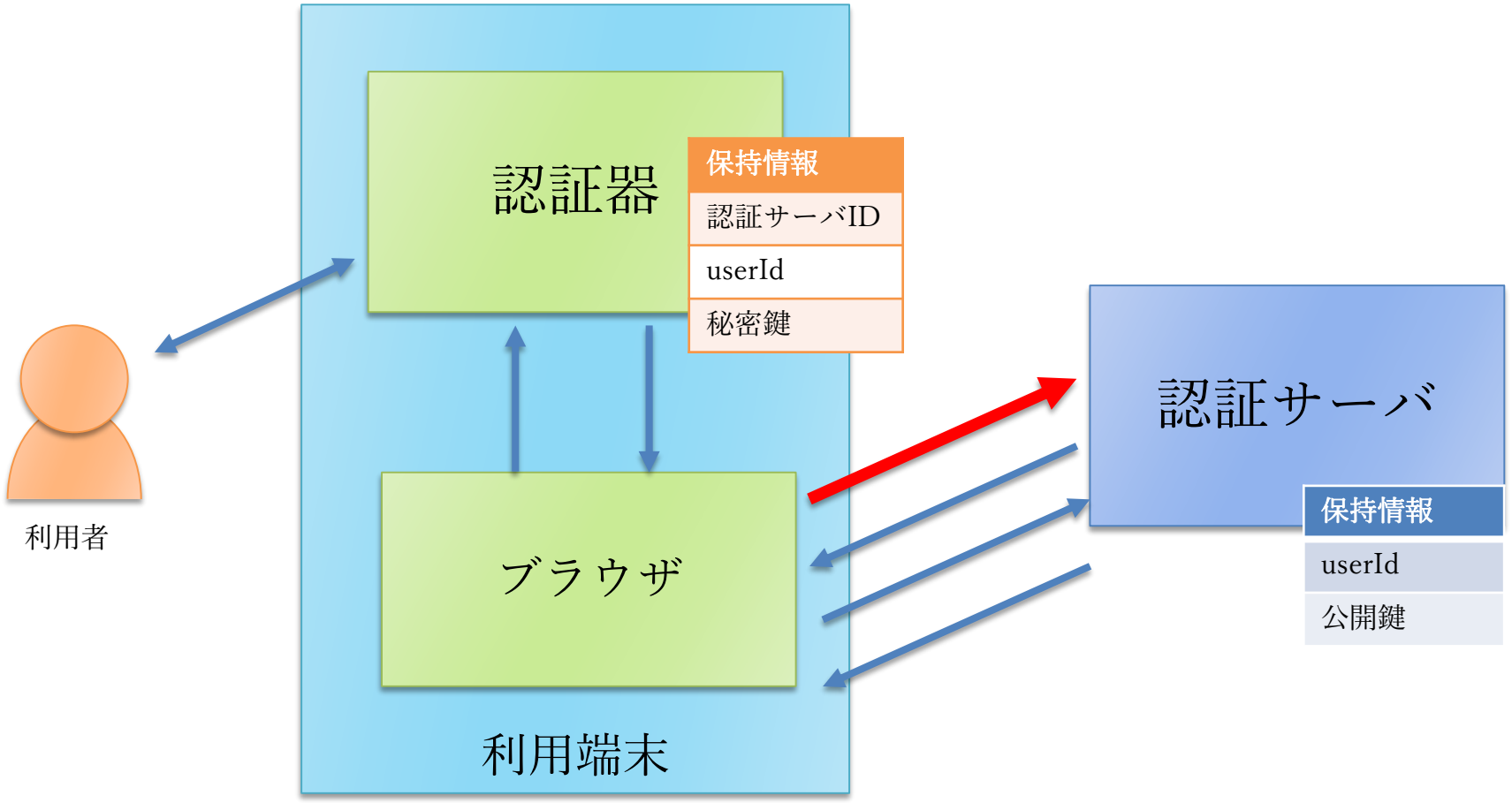
- **これらの内複数の要素を満たす認証を「多要素認証」と呼ぶ**
  - パスワード＋生年月日やパスワード＋秘密の質問では「多要素認証」ではなく同一要素の「多段階認証」である
- **各認証要素には前述の特徴があり、複数の要素を組み合わせることで欠点を補いセキュリティレベルを高めることができる**
  - PWと秘密の質問で多段階認証にしても、ユーザが他サイトと同一の組み合わせを利用していたら他サイト側での漏洩により突破される可能性も
  - PWとクライアント証明書による多要素認証であれば一方が漏洩、紛失してももう一方を攻撃者が取得・推測することは難しい

FIDO利用時に多要素認証を実現する方法

- **FIDO単体では公開鍵認証（＝秘密鍵を持っている）ことによるWYH要素を満たしている**
  - 認証器側で生体認証やPINによる認証を追加することで多要素認証とできる
  - 認証器側にユーザ認証機能がない場合、認証サーバ側でFIDO＋パスワード認証を課すことも可能
    - 1要素目にパスワードによるWYK
    - 2要素目にFIDOのWYHを用いる
- **FIDOの利用＝パスワードレス認証ではなく、FIDO（公開鍵認証）＋認証器での生体認証によって多要素パスワードレス認証は実現される**

動作概要

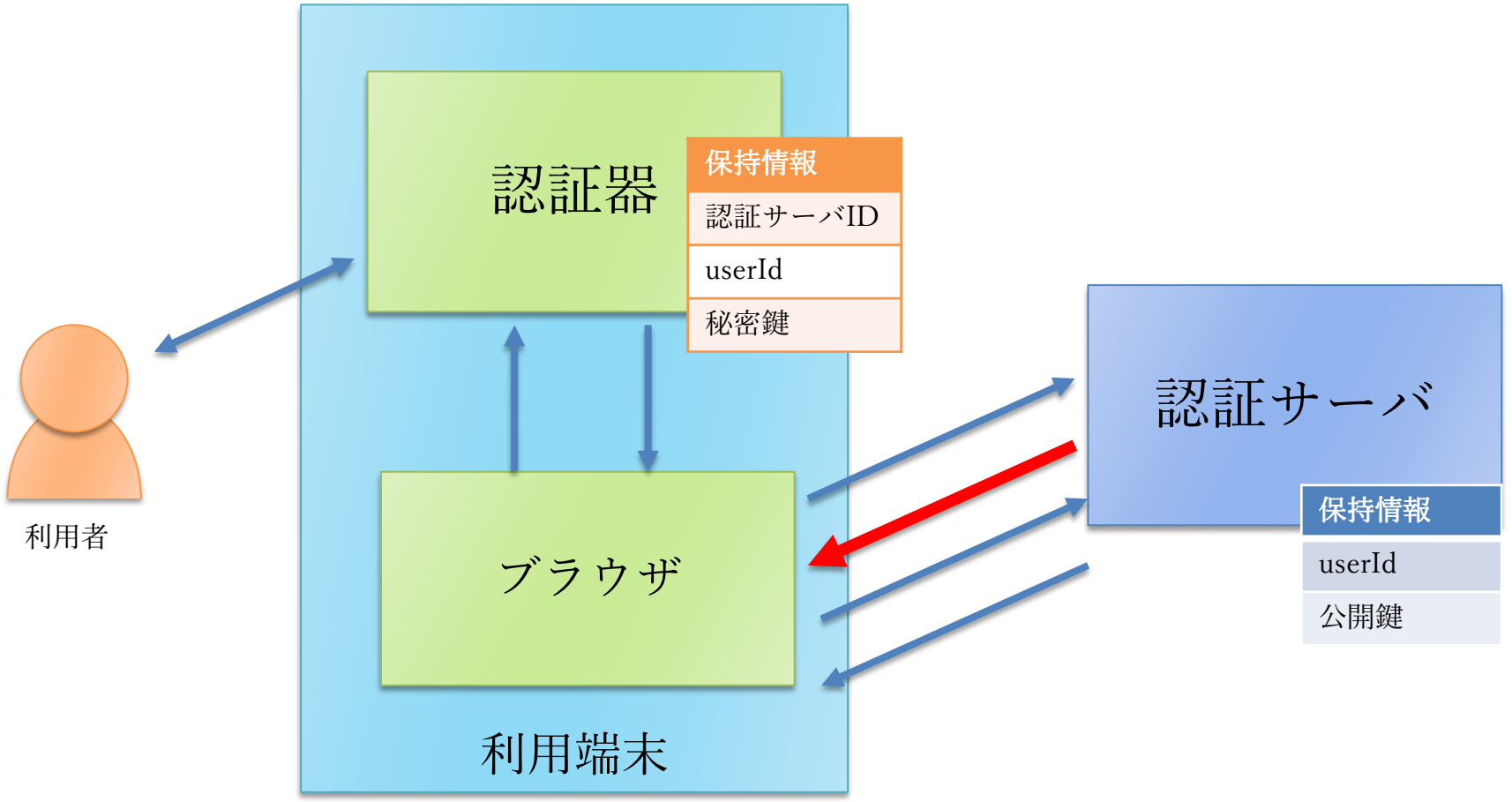
□ 認証



1. ログイン処理の開始
2. challengeの返却
3. 署名要求
4. ユーザ認証
5. 署名後challenge返却
6. 署名後challenge送信
7. 署名の検証

動作概要

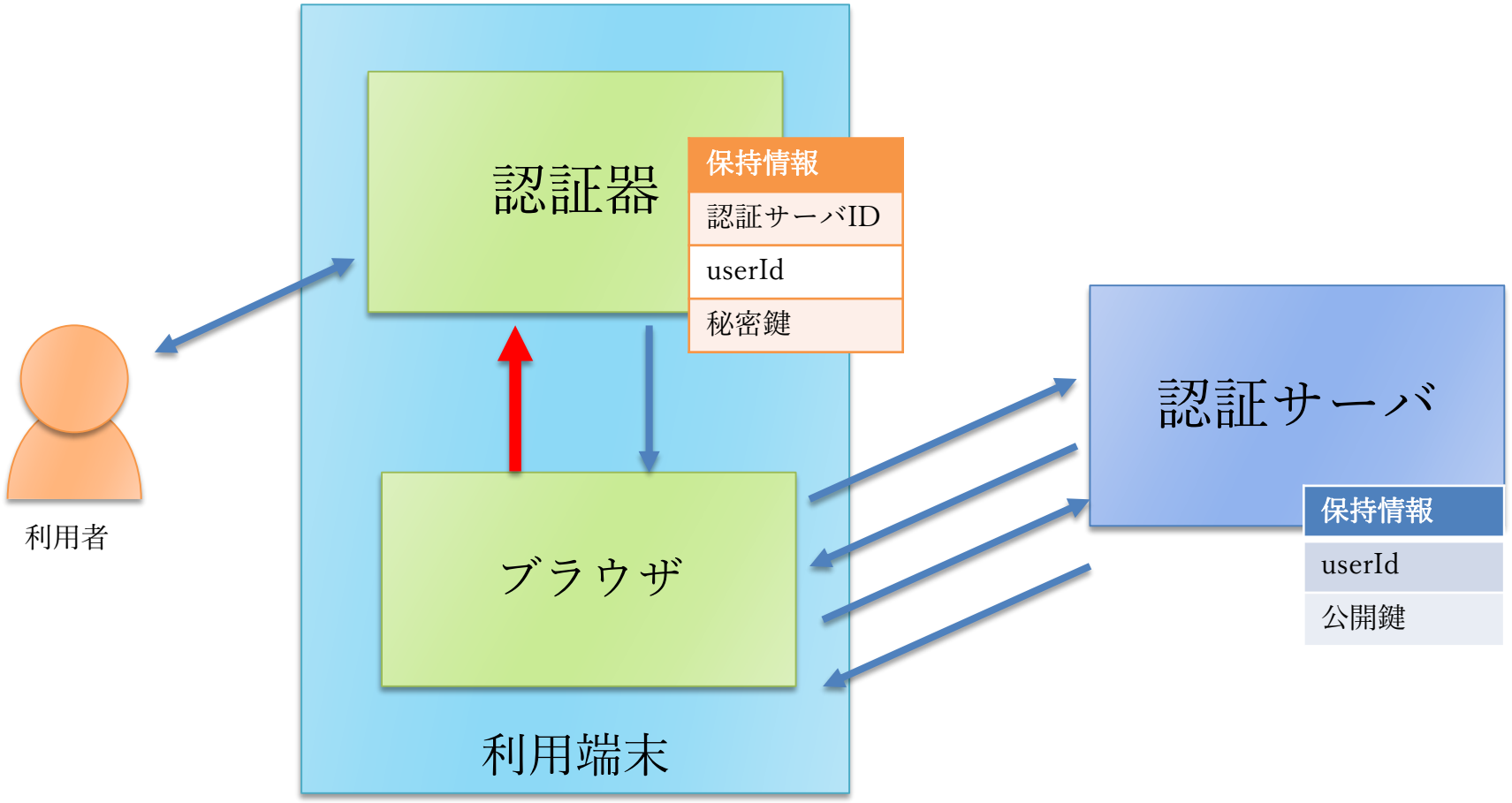
□ 認証



1. ログイン処理の開始
2. challengeの返却
3. 署名要求
4. ユーザ認証
5. 署名後challenge返却
6. 署名後challenge送信
7. 署名の検証

動作概要

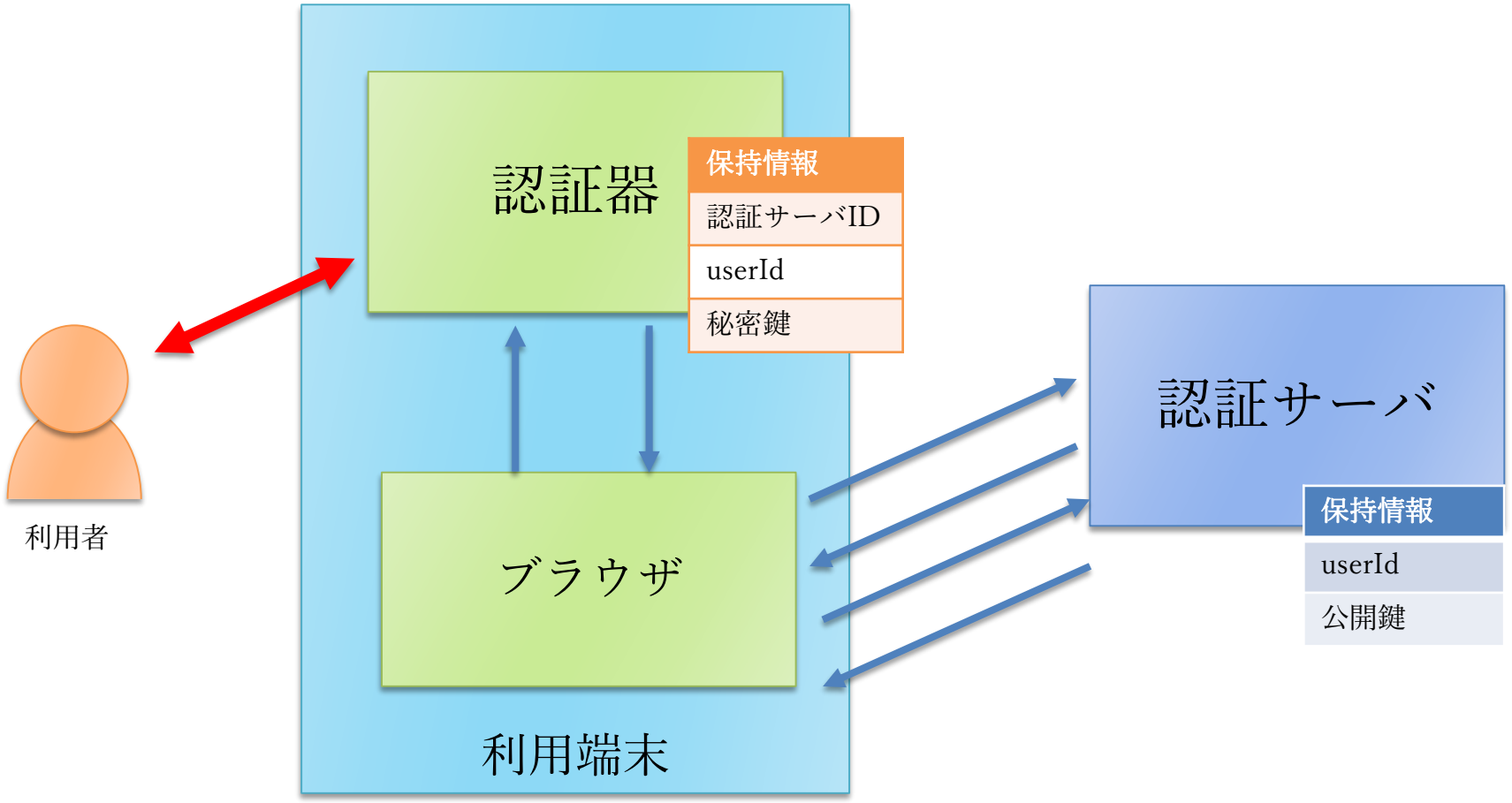
□ 認証



1. ログイン処理の開始
2. challengeの返却
3. 署名要求
4. ユーザ認証
5. 署名後challenge返却
6. 署名後challenge送信
7. 署名の検証

動作概要

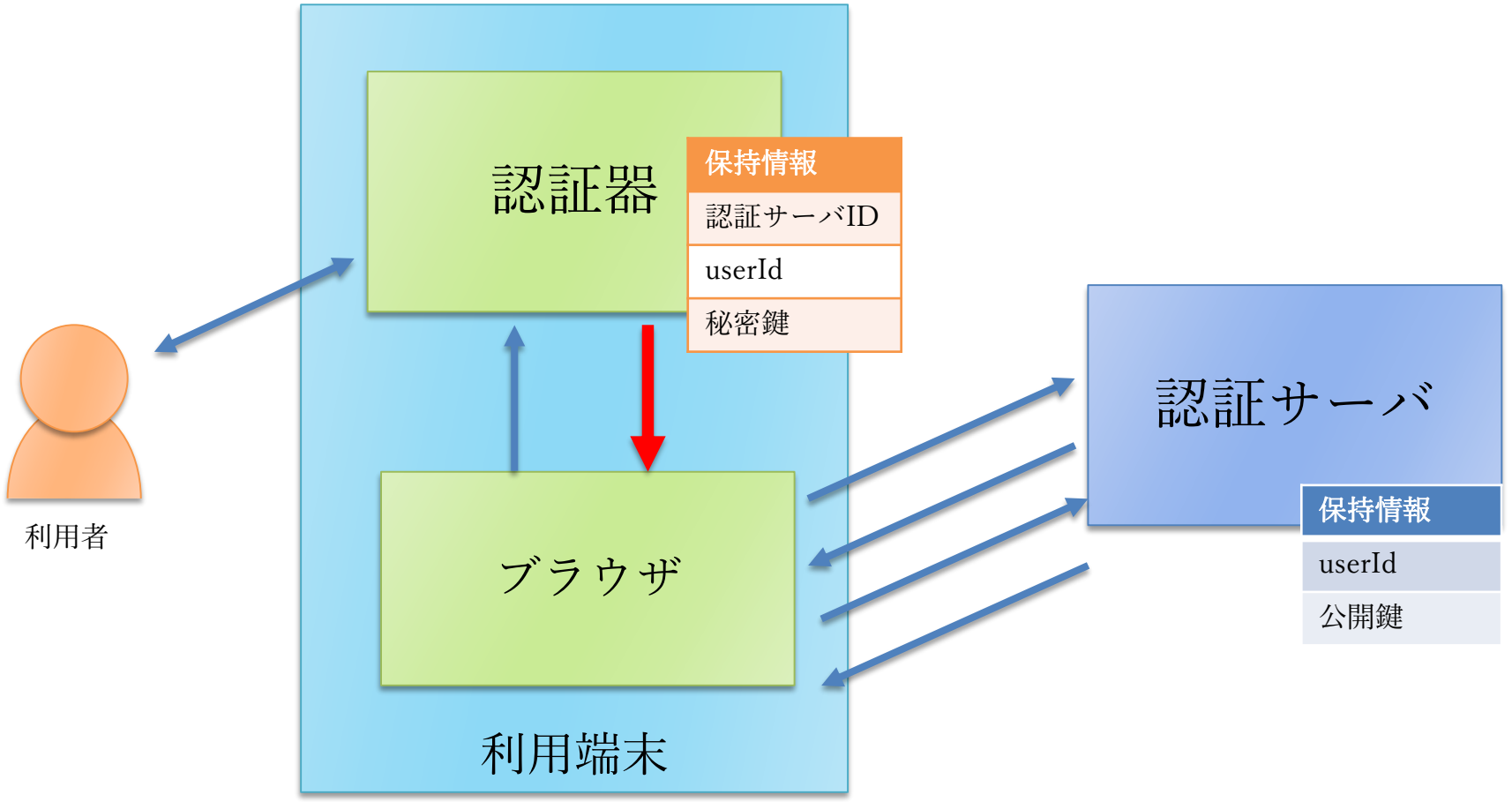
□ 認証



1. ログイン処理の開始
2. challengeの返却
3. 署名要求
4. ユーザ認証
5. 署名後challenge返却
6. 署名後challenge送信
7. 署名の検証

動作概要

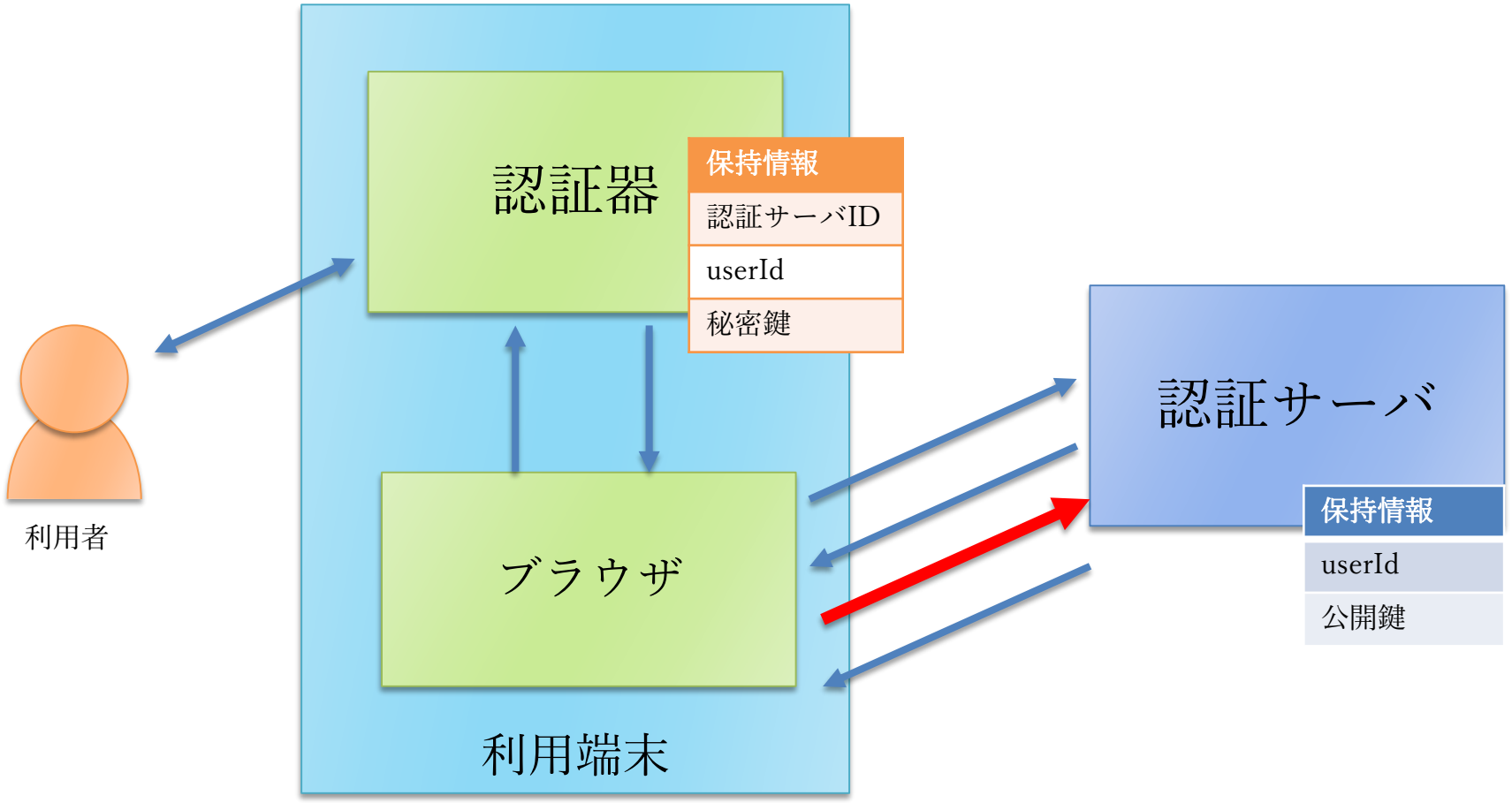
□ 認証



1. ログイン処理の開始
2. challengeの返却
3. 署名要求
4. ユーザ認証
5. 署名後challenge返却
6. 署名後challenge送信
7. 署名の検証

動作概要

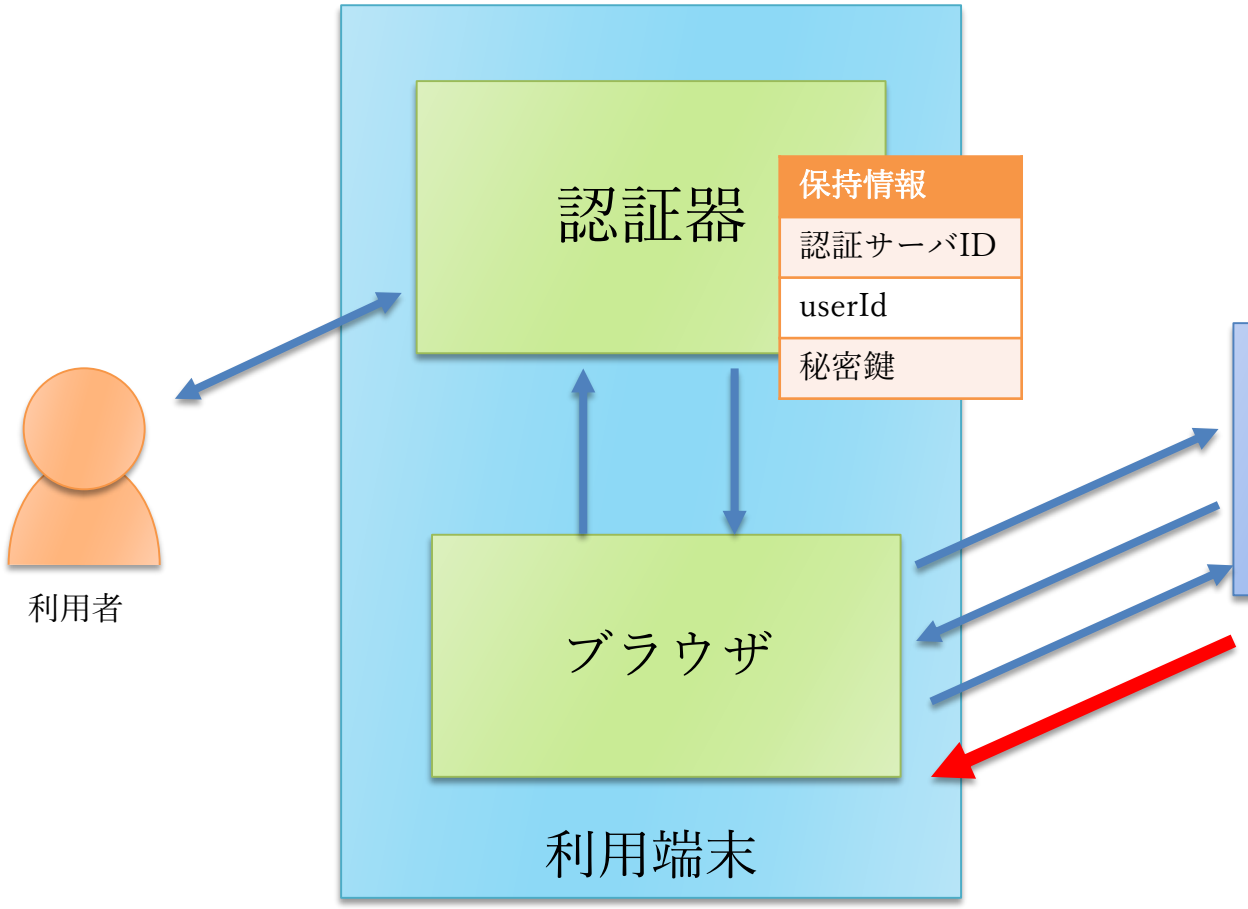
□ 認証



1. ログイン処理の開始
2. challengeの返却
3. 署名要求
4. ユーザ認証
5. 署名後challenge返却
6. 署名後challenge送信
7. 署名の検証

動作概要

□ 認証



1. ログイン処理の開始
2. challengeの返却
3. 署名要求
4. ユーザ認証
5. 署名後challenge返却
6. 署名後challenge送信
7. 署名の検証

今後の課題

- **FIDOではデバイス内に秘密鍵を保管するため、内蔵認証器を利用する場合にどのように他端末によるログインを実施させるか**
  - 例：スマートフォンでアカウント登録した場合に、PCでログインするときの方法
    - Bluetooth接続などが検討されている
    - デスクトップPCで登録し、他拠点のデスクトップPCでアカウントを利用したい場合はどうすればいいか
- **デバイスの紛失した場合のアカウントリカバリ**
  - 既存の鍵を無効化し、新端末での利用をどのように開始するか

上記課題を解決できれば、さらなる普及が進むのではないかと予想される

CIBA
(OpenID Connect Client Initiated Backchannel Authentication Flow)

CIBA (Client Initiated Backchannel Authentication)

- OpenID Foundationが策定を進めるドラフト仕様の一つ
- OpenID Connectではブラウザを介して認証基盤とアプリケーション間で認証情報を受け渡すが、CIBAはサーバ間通信で認証情報を受け取るための仕様
- アプリケーションを操作している人ではなく第三者の認証情報を確認する際に利用する

ユースケース

□ 遠隔地のユーザを認証することができる

- コールセンターへの架電を行ったユーザが本人であることを認証する

□ 対面のユーザを認証することにも用いることができる

- 窓口対応しているユーザが身分証明書を出さずとも本人であることを認証する
- クレジットカード決済時に決済端末にPIN入力せず、利用者のスマートフォンで認証する

CIBAのメリット

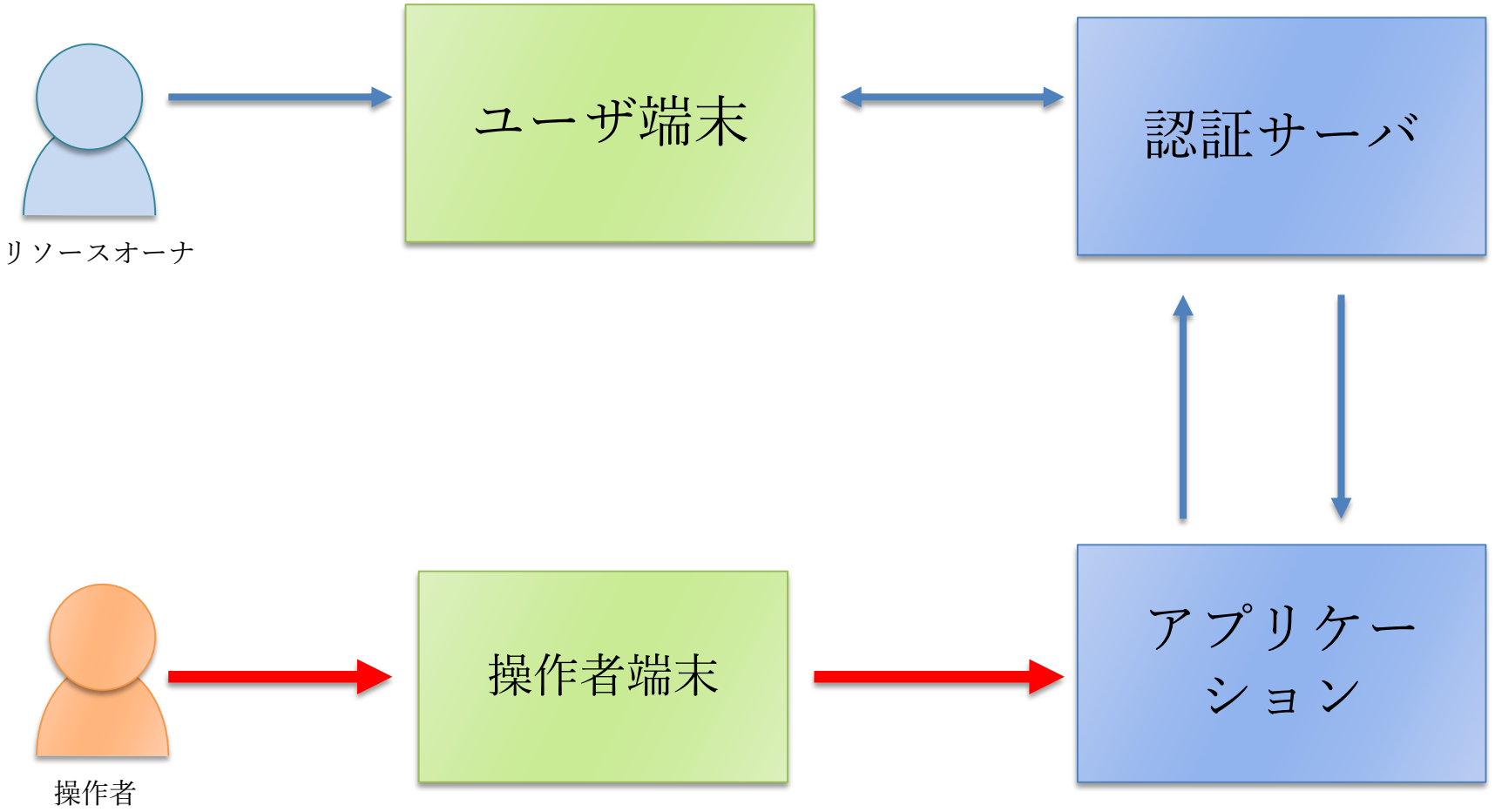
ロ ユーザが他者の端末にクレデンシャルを入力する必要がない

- 他者端末からのクレデンシャル漏洩リスク・傍受リスクを低減できる

ロ 遠隔地のユーザ認証を認証する際のセキュリティレベル向上

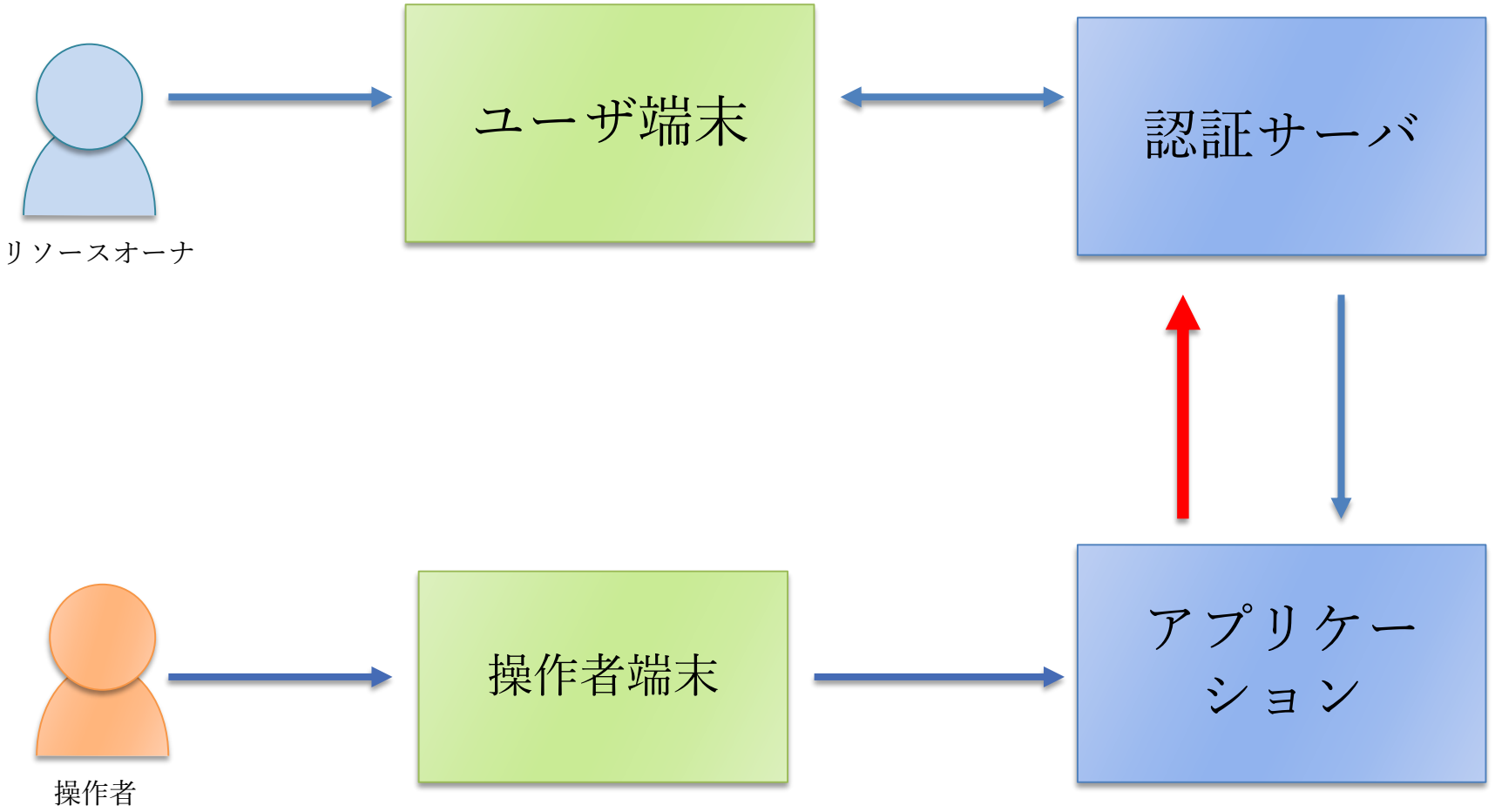
- 従来、遠隔地ユーザの本人性を確認する際、生年月日・氏名等の個人情報を聞き出すことでユーザの特定を行っていたが、WYK（知っている）要素での認証が主だった。
  - CIBAを利用すると安全に、多要素にも対応可能

動作概要



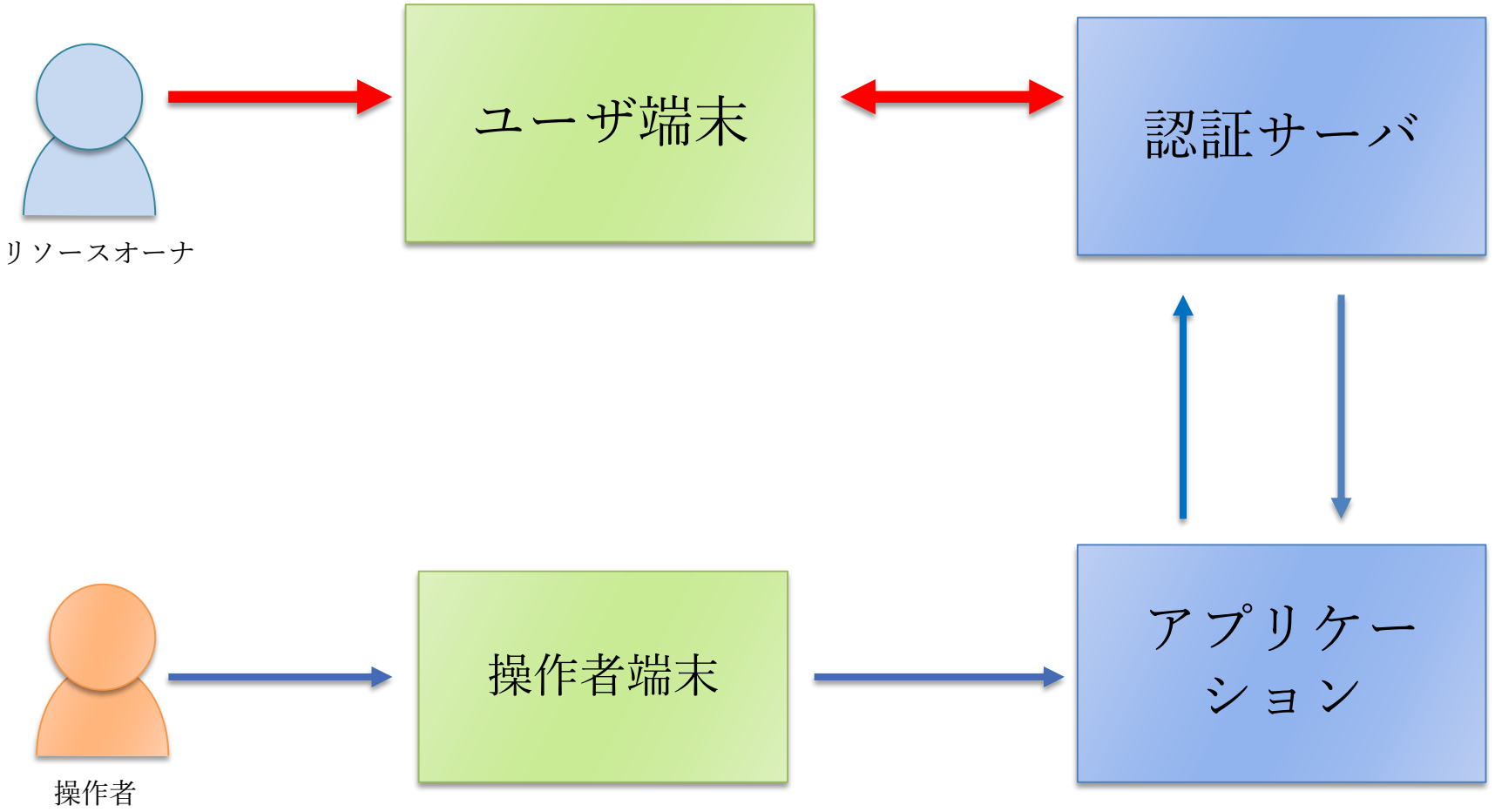
1. リソースオーナを特定する情報の入力
2. 認証要求
3. ユーザ認証
4. 認証情報の返却

動作概要



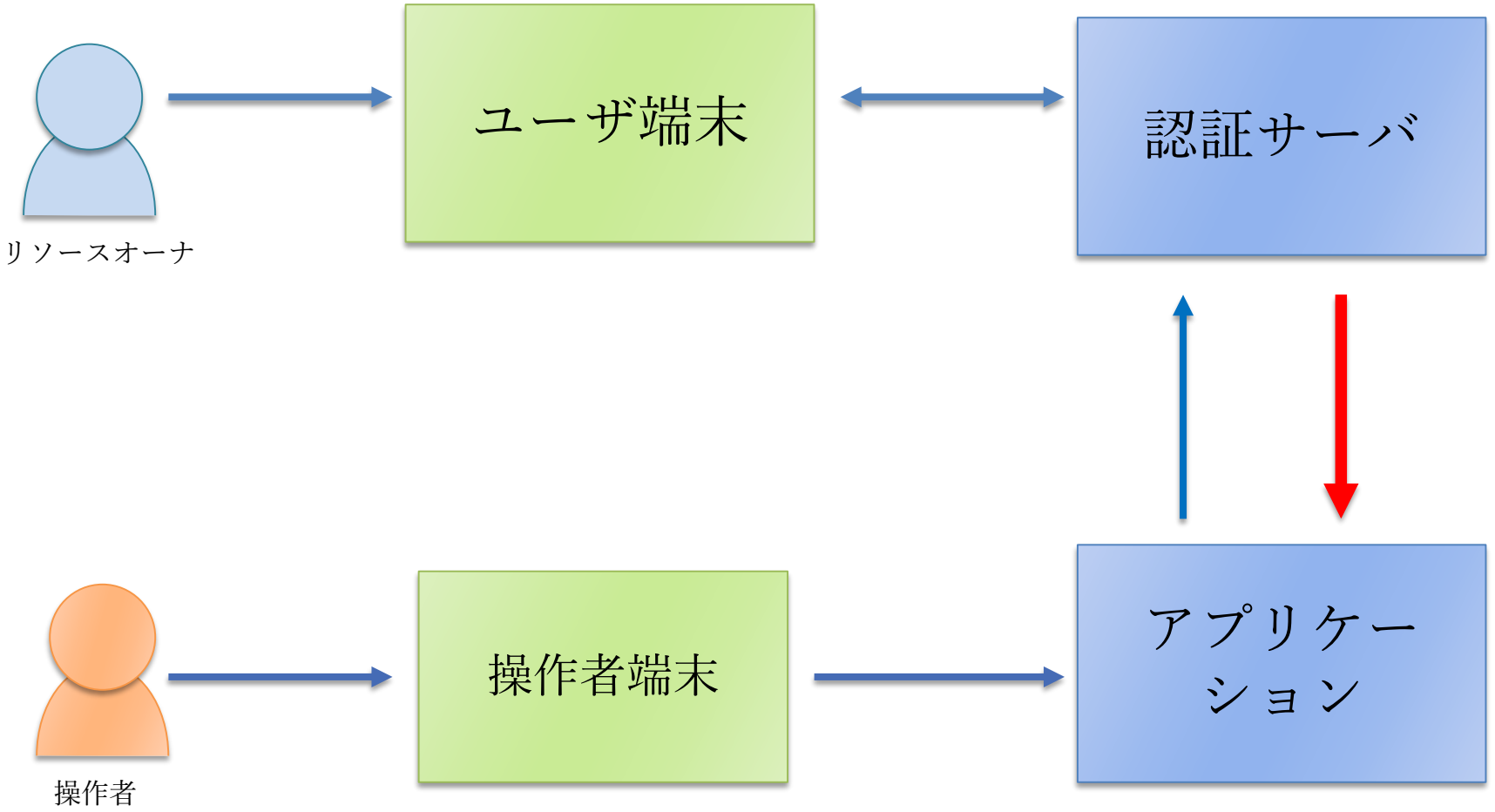
1. リソースオーナを特定する情報の入力
2. 認証要求
3. ユーザ認証
4. 認証情報の返却

動作概要



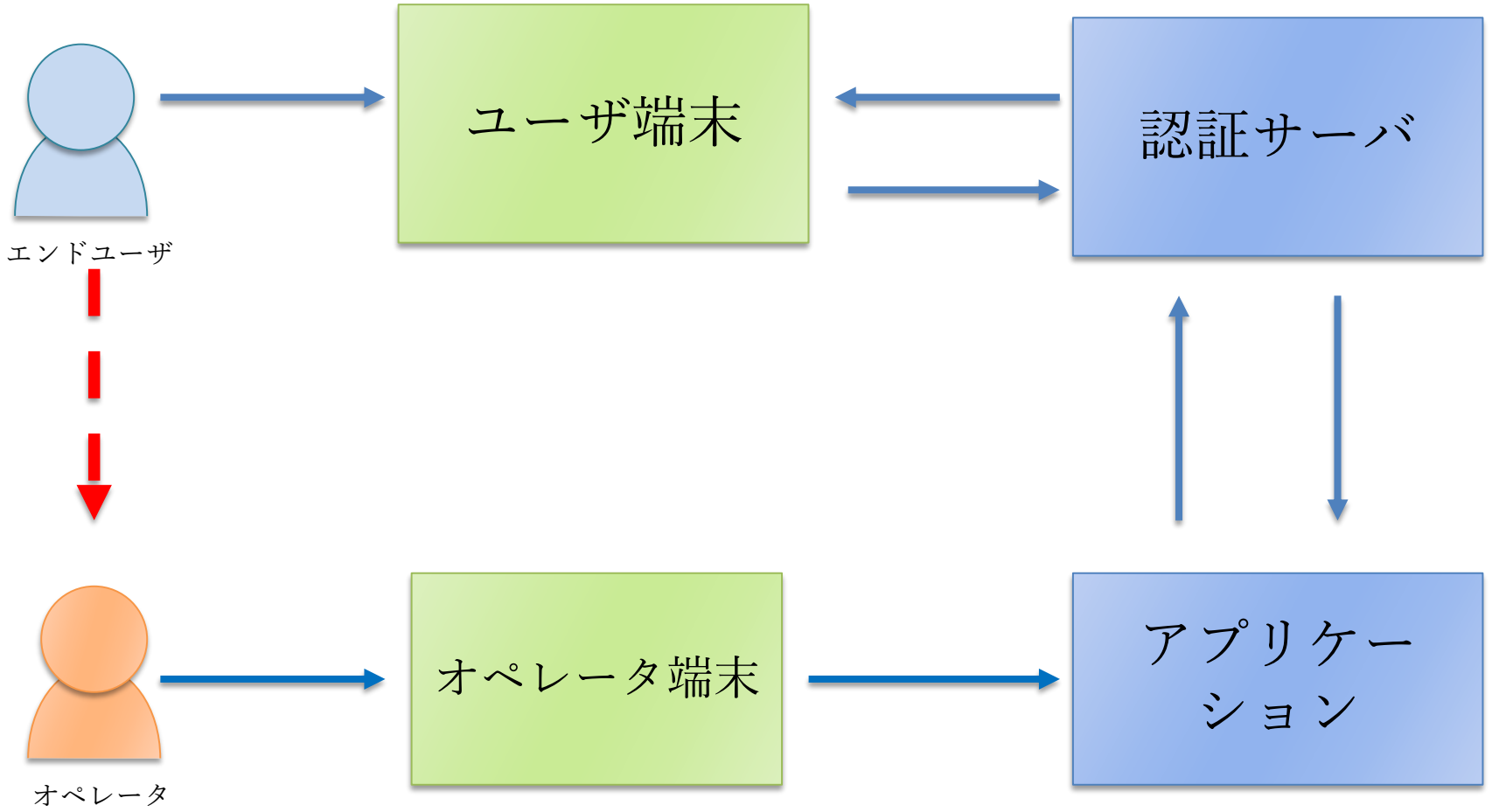
1. リソースオーナを特定する情報の入力
2. 認証要求
3. ユーザ認証
4. 認証情報の返却

動作概要



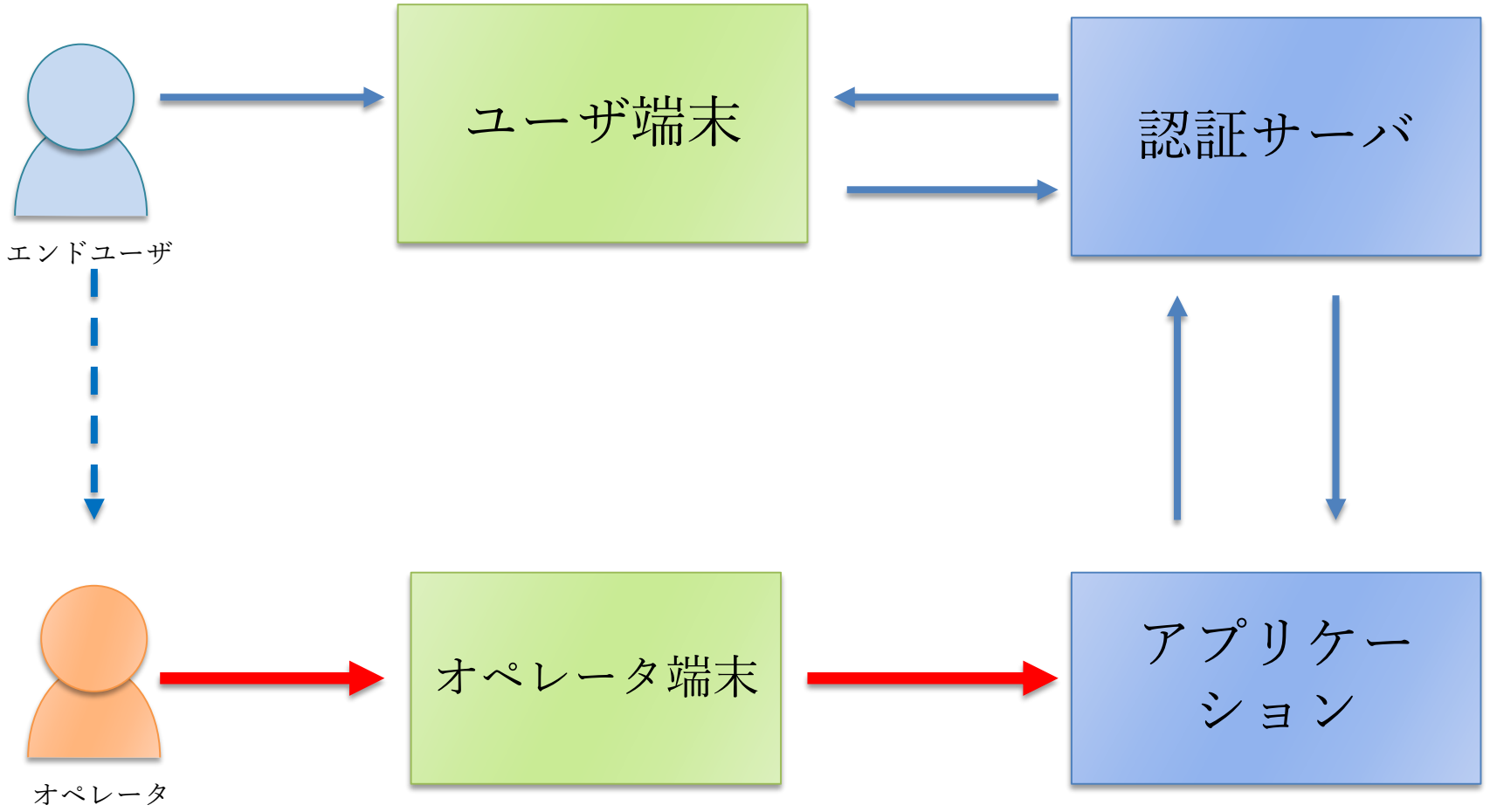
1. リソースオーナを特定する情報の入力
2. 認証要求
3. ユーザ認証
4. 認証情報の返却

コールセンターでエンドユーザの認証を行う



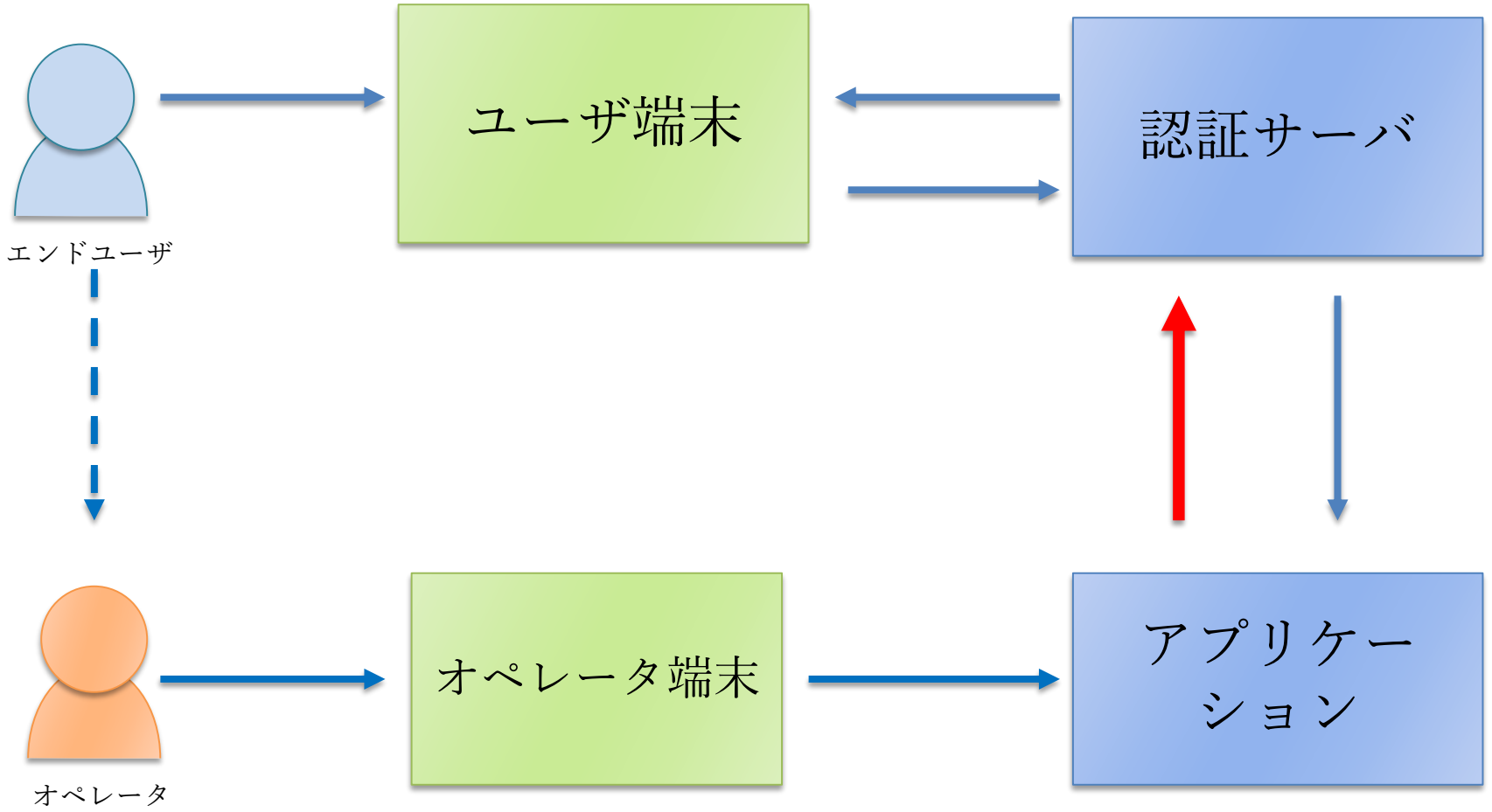
1. ユーザがコールセンターへ電話
2. ユーザの電話番号入力
3. 認証要求
4. ログインURLをメール送信
5. ユーザ認証
6. 認証情報の返却

コールセンターでエンドユーザの認証を行う



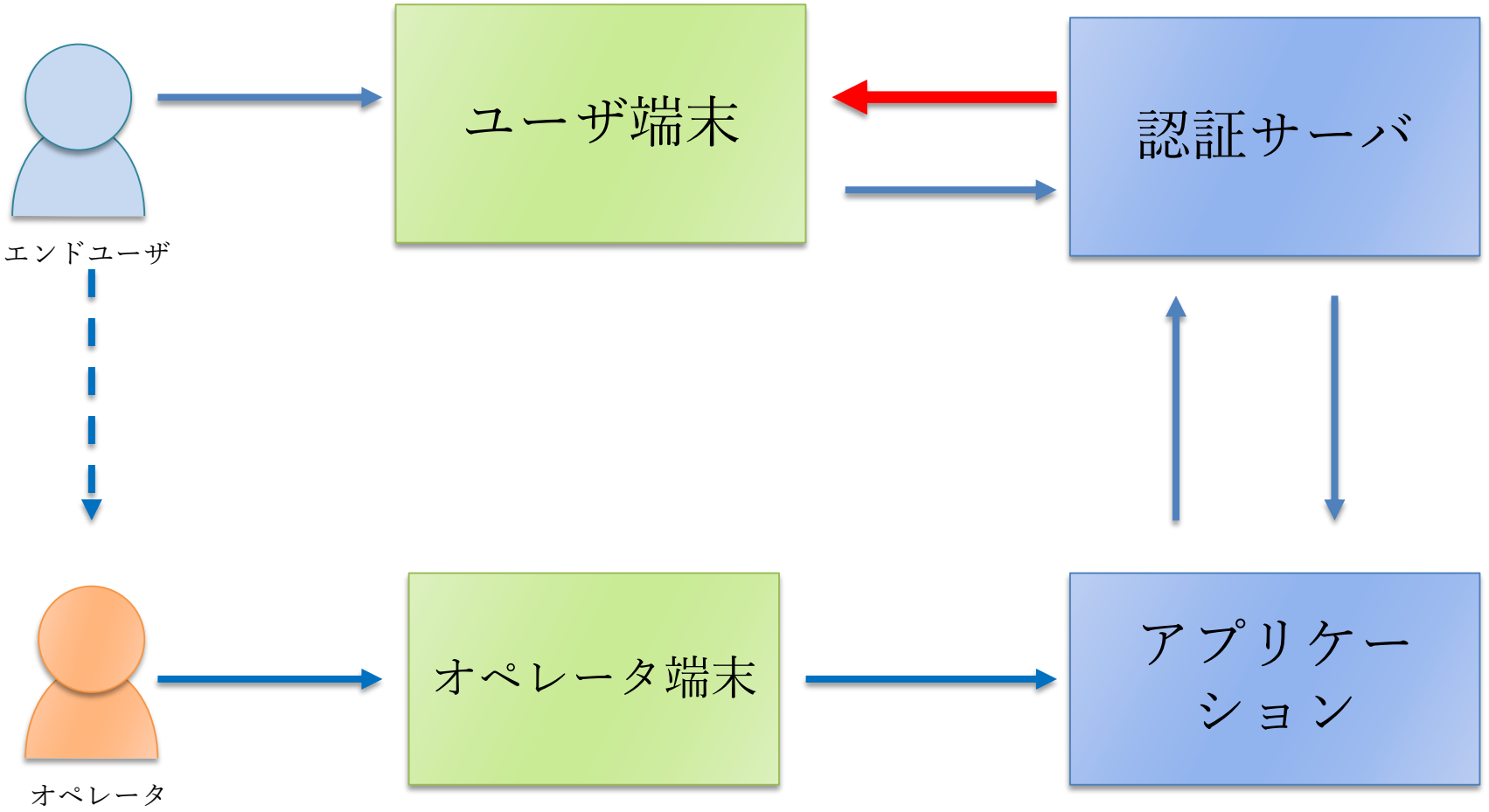
1. ユーザがコールセンターへ電話
2. ユーザの電話番号入力
3. 認証要求
4. ログインURLをメール送信
5. ユーザ認証
6. 認証情報の返却

コールセンターでエンドユーザの認証を行う



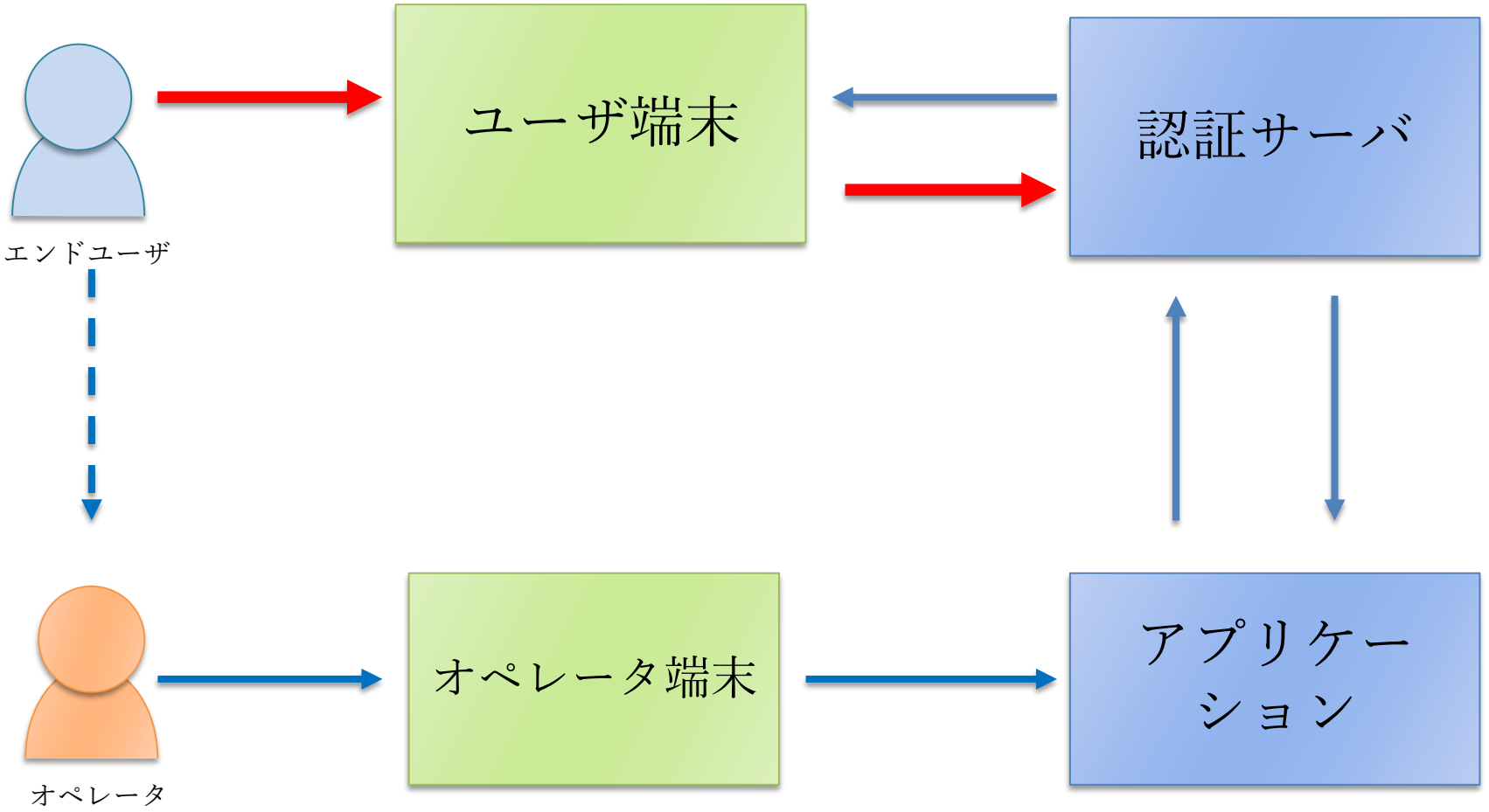
1. ユーザがコールセンターへ電話
2. ユーザの電話番号入力
3. 認証要求
4. ログインURLをメール送信
5. ユーザ認証
6. 認証情報の返却

コールセンターでエンドユーザの認証を行う



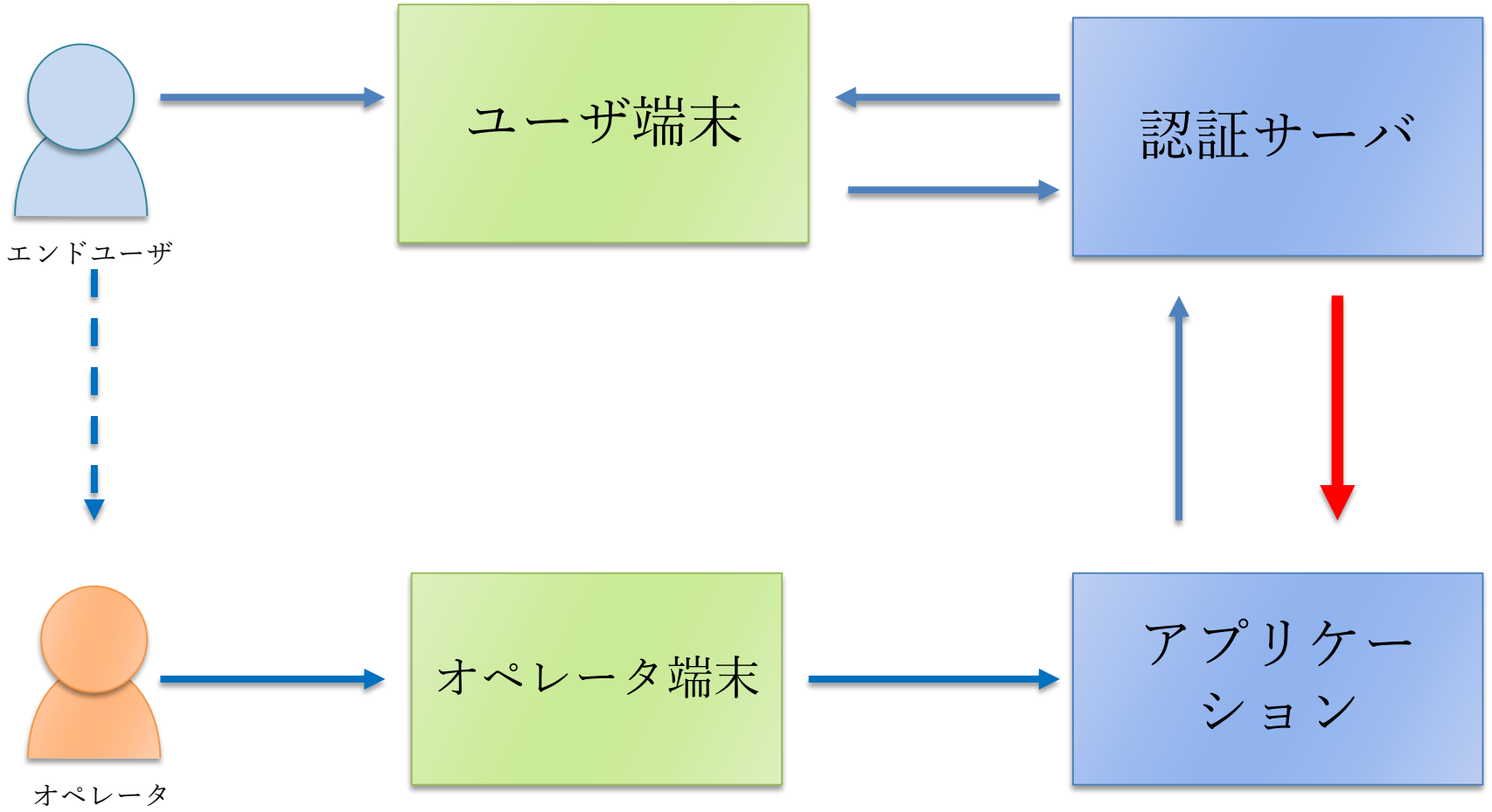
1. ユーザがコールセンターへ電話
2. ユーザの電話番号入力
3. 認証要求
4. ログインURLをメール送信
5. ユーザ認証
6. 認証情報の返却

コールセンターでエンドユーザの認証を行う



1. ユーザがコールセンターへ電話
2. ユーザの電話番号入力
3. 認証要求
4. ログインURLをメール送信
5. ユーザ認証
6. 認証情報の返却

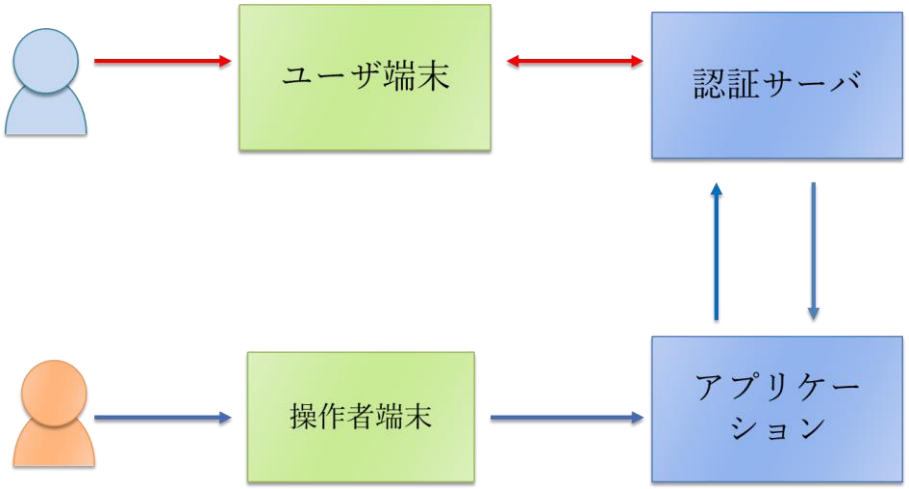
コールセンターでエンドユーザの認証を行う



1. ユーザがコールセンターへ電話
2. ユーザの電話番号入力
3. 認証要求
4. ログインURLをメール送信
5. ユーザ認証
6. 認証情報の返却

認証系標準技術の組み合わせ

- CIBAとFIDOでは適用範囲が異なる仕様であるため、組み合わせることも可能
  - CIBAは認証サーバとアプリケーション間の認証連携を規定している
  - FIDOは認証サーバとユーザ間の認証方法を規定している



まとめ

まとめ

- **システムを構成するにあたり、標準技術を知ることが重要である**
- **常に新しい仕様に目を向けることが必要である**
  - 標準技術は危殆化し、代替技術や追加仕様が策定される
  - 新しい標準技術には利便性・セキュリティ向上を行える可能性がある
- **一方、既存の標準技術を学ぶこと・最新動向を追い続けることは高い学習コストが必要になる**
  - 難しい場合は標準仕様に沿ったソリューションの導入を検討したほうが良い

ご清聴ありがとうございました